CYCLIC PROJECT SCHEDULING

A THESIS

Presented to

The Faculty of the Division of Graduate

Studies and Research

By

David Russell Hickman

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Industrial Engineering

Georgia Institute of Technology

August 1974

CYCLIC PROJECT SCHEDULING

Approved:

Richard H. Deane, Chairman

R. Gary Parker

Russell G. Heikes

Date approved by Chairman: 8/9/74

ACKNOWLEDGMENTS

I would like to express my appreciation to Dr. Richard Deane, my thesis advisor, for his valuable help in the preparation of this thesis. His continuing interest in my work contributed significantly to its successful completion.

I would also like to express my thanks to Dr. Gary Parker and Dr. Russ Heikes for serving on my reading committee and for their useful comments and advice. I also should thank both of them for their contributions in the classroom, as well as on this thesis, which have helped to make my graduate education worthwhile.

Thanks are also expressed to Dr. W. W. Hines for his support and his interest in my graduate education.

Finally, I would like to dedicate this thesis to my wife, Judy, for without her patience and hard work, it would have never been written.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

SUMMARY

This study concerns the scheduling of parallel multi-cycle projects using network scheduling techniques. The major objective of the scheduling techniques which are developed is to reduce costs or other penalties resulting from interruptions between cycles of activities.

Several objective functions are defined and are applied to a modified version of a previously developed scheduling algorithm. This modified algorithm reduces interruption costs by reducing both the number and the lengths of interruptions in the project schedule. Under the assumption that the project completion date may be extended at some known cost, the scheduling algorithm may be extended to cover situations where selected activities must be continuously scheduled or where scheduling for arbitrarily selected completion times is desired. A time-cost trade-off procedure is also developed, where a low cost schedule is obtained by extending the project completion time so that interruption costs may be reduced or eliminated.

Computer programs to perform some of the scheduling computations are presented to demonstrate the capability of developing computerized scheduling systems for cyclic projects. An extensive application of the time-cost trade-off procedure to a sample problem is also given.

CHAPTER I

INTRODUCTION

The Program Evaluation and Review Technique (PERT) and the Critical Path Method (CPM) have had a significant impact on the planning, scheduling, and control of large-scale projects. During the past fifteen years, each has become recognized as a valuable aid to project management.

PERT is an event-oriented network-based scheduling technique that was developed jointly in the late 1950's by the Special Projects Office of the United States Navy, the Lockheed Aircraft Corporation, and the consulting firm of Booz, Allen, and Hamilton. Its development was necessitated by a desire of the Navy to efficiently control the development of the Polaris missile program. PERT utilizes multiple time estimates for the duration of activities in the project and provides a statement of the probabilities of achieving various completion times for the project.

CPM was developed in 1957 by E. I. du Pont de Nemours and Co. and the Univac division of Remington Rand. Though this was simultaneous to the development of PERT, the development of CPM was completely independent and unknown to those working with the Navy. The heart of CPM is the use of network diagrams in which directed arcs represent the activities of the project. Deterministic activity duration times are utilized to determine the schedule and completion time of a project. Originally developed to allow du Pont to achieve more efficient scheduling of plant and maintenance functions, CPM has now found wide applicability, particularly in the building construction industry.

Following the initial development of CPM and PERT, there was a significant amount of research undertaken to extend their application to solving a wide variety of problems. With respect to CPM, much of this research has focused on the development of techniques for cost reduction and time-cost trade-offs, leveling of applied resources, and efficient allocation of limited resources. However, two other aspects of project scheduling--cyclic projects and continuity of activities--have received very little attention in past research, although each appears to be very important in the scheduling of certain types of projects that occur frequently in industry.

Description of Cyclic Projects

A cyclic project is one in which the project, or at least a significant portion of it, can be described as the repetition of a set of activities that define distinctive identical phases of work. In the case where only a portion of a project is cyclical, the methods developed in this research will apply only to the cyclic portion, which will also be referred to as a "cyclic project."

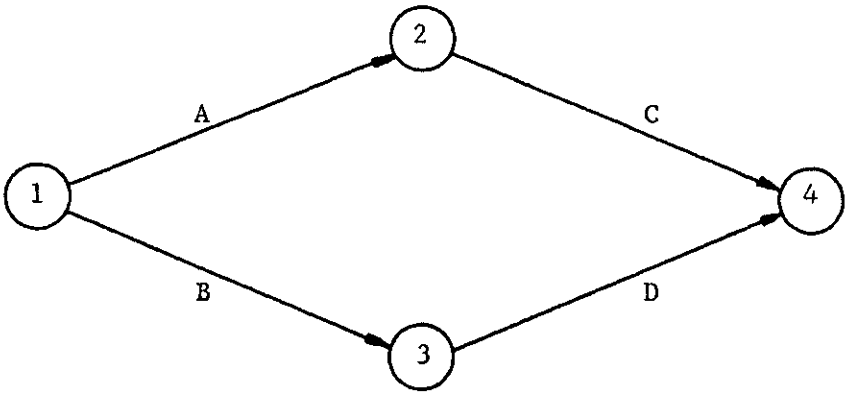
As an example of a cyclic project, consider the construction of a multi-story hotel or office building. Since the floors of many of these structures are identical, the construction of one floor can be considered to be one cycle of the project. The number of cycles in the project would be represented by the number of floors in the structure. Thus, to build a sixty floor hotel, it would only be necessary to describe the construction of one floor and to specify that this cycle be repeated sixty times. To apply CPM scheduling techniques, however, requires further clarification

of the relationships between the activities in succeeding cycles of the project. As an example of the two different methods of planning and scheduling of cyclic projects, consider the project networks presented in Figure 1.

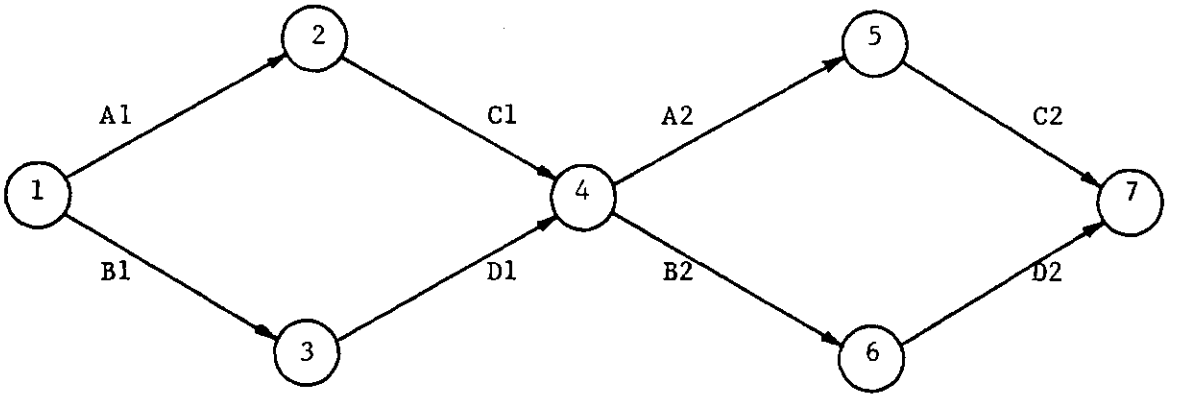
Suppose part (a) of Figure 1 represents a single cycle of a simple project and it is required that two cycles of the activities described by this network be completed. One possible interface of the two cycles is for the terminal node of the first cycle to be the initial node of the second cycle: i.e., the second cycle is begun only when the first cycle of all activities has been completed. This form is represented by the network of part (b) in Figure 1. Such a network is called a serial multi-cycle project network because the total project is simply the repetition of identical single cycles connected in series.

Common sense and an understanding of project scheduling indicates, however, that a serially scheduled cyclic project often does not represent the most realistic conception of how such a project is actually carried out. For example, in constructing a multi-story building, it would never be necessary to complete the construction of an entire floor before beginning the construction of the next floor. It is usual practice in CPM scheduling to schedule the beginning of an activity as soon as all of its predecessors have been completed. Then, an activity in cycle two of the project has as its predecessors only the activities which precede it in cycle two of the project plus the first cycle of the same activity. Therefore, these precedence relationships would result in a network like that of part (c) of Figure 1. This network represents what is known as a

(a) One cycle of a project



(b) Two cycles of (a) as a serial multi-cycle project



(c) Two cycles of (a) as a parallel multi-cycle project

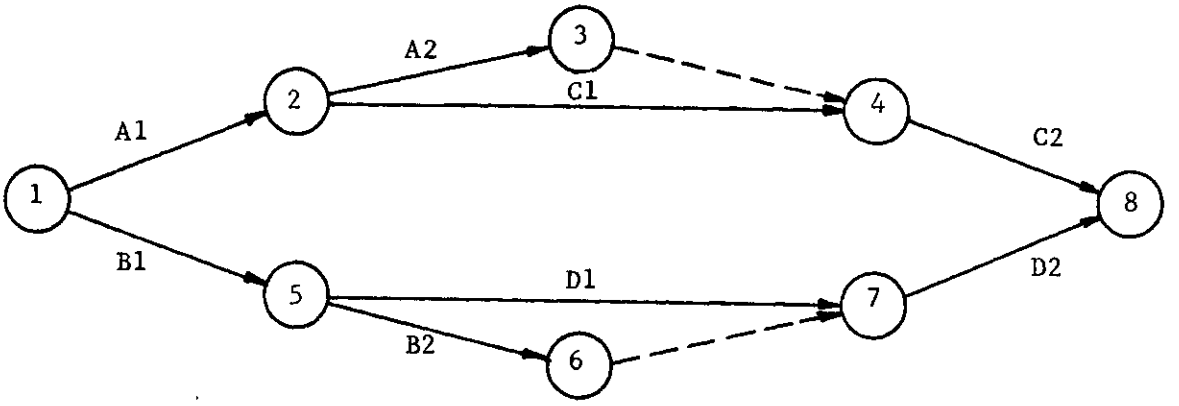


Figure 1. Examples of Cyclic Projects

parallel multi-cycle project network because such a network can be thought of as a succession of identical networks stacked one upon the other with interconnecting dummy activities between cycles of identical activities. A representation of this concept for an n-cycle project based upon part (a) of Figure 1 is shown in the diagram of Figure 2.

Though parallel networks appear to be fairly complex, their properties actually allow for very easy application of the CPM computations to determine the early and late start and finish times and the activities on the critical path. However, the repetition of activities also results in very special problems involving activity continuity.

Description of Activity Continuity

It is often desirable or even necessary that the completion of one activity be followed immediately by the beginning of a successor activity. This creates a problem of continuity of activities when scheduling a project.

Burman (3) has defined two different types of activity continuity in project scheduling: close continuity and loose continuity. Close continuity implies that the start of one activity must follow immediately upon the completion of a preceding activity. Loose continuity implies that the start of one activity must follow within some stipulated time period after the completion of a predecessor activity.

Continuity is not at all a problem when there is only one predecessor and successor activity. However, multiple predecessor and/or successor relationships preclude the possibility of continuity of some activity schedules if the minimum completion time of a project is required. When considered in the scheduling of parallel multi-cycle projects, the

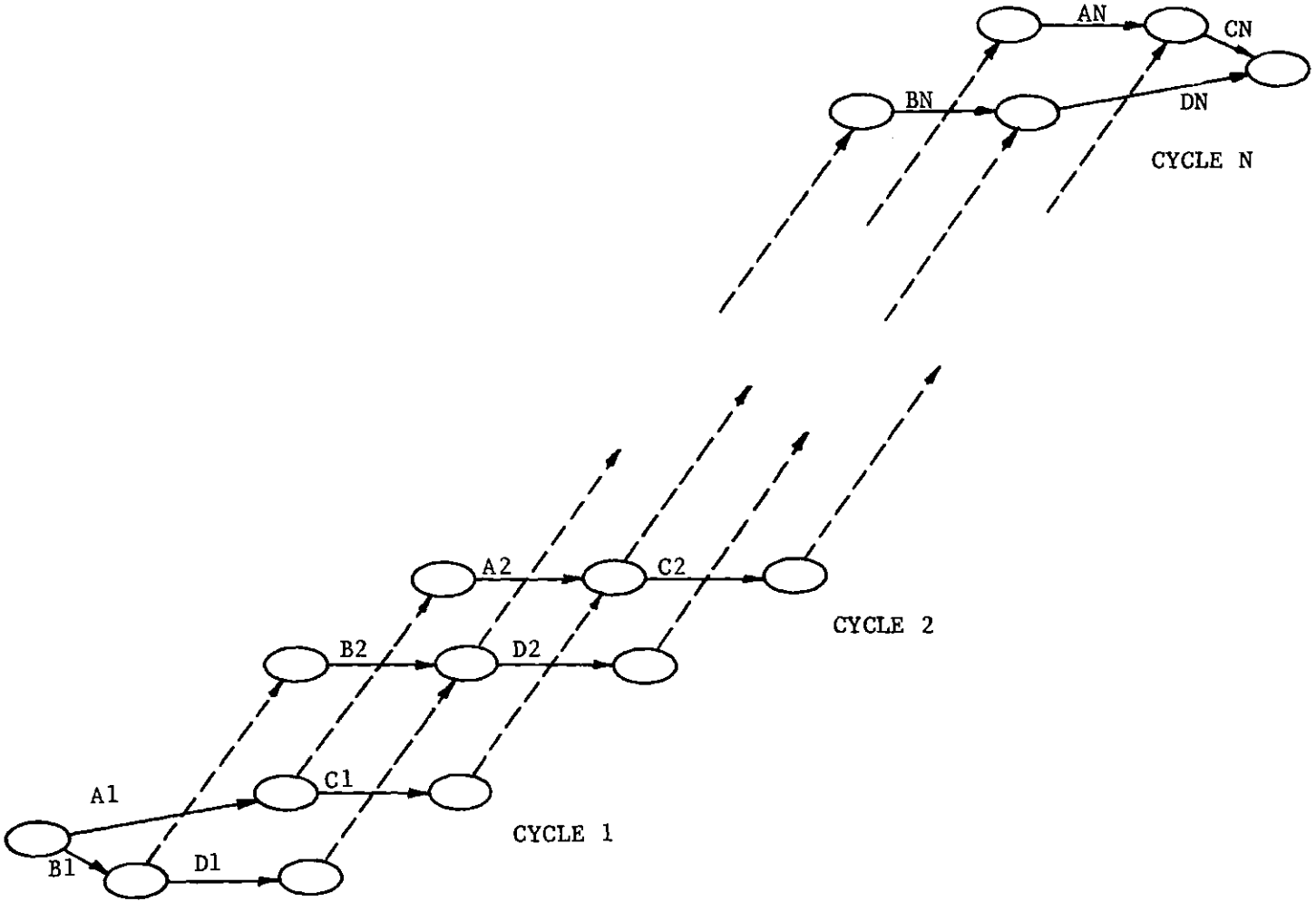


Figure 2. Three Dimensional Representation of a Parallel Multi-Cycle Project

idea of activity continuity can result in some very important and complex problems.

Description of the Problem

Because of the repetition of identical activities, continuity will be more important in multi-cycle projects than in non-repetitive ones. If unnecessary interruptions between cycles of activities are allowed to occur, avoidable costs and inefficiencies can result. Some of these would be due to: (1) recurrence of set-up and shut-down costs; (2) idle manpower, if it cannot be utilized elsewhere; (3) costs of maintaining idle equipment, if it cannot be utilized elsewhere; (4) hiring and firing costs, if interruptions are of sufficient length to prevent keeping specialized employees on the payroll; and (5) diminished beneficial effects of the learning factor. If some of the cycles of activities could be scheduled without interruption, many of these could be avoided.

The example provided by Figures 3 and 4 shows how significant these inefficiencies can be. Figure 3 depicts a network with twenty-eight activities that are to be repeated for five cycles. The numbers above the arcs on the diagram represent the duration of each activity. The data from this network was used in a standard CPM program which calculated the early start time for each cycle of every activity. This schedule is depicted in a Gantt chart in Figure 4. The schedule for all cycles of one activity is shown on a single line. The numbers on each line are used to form the bars which represent the duration of the activity's cycle; and the number forming the bar is the number of the cycle.

It is important to note in the schedule of Figure 4 that almost

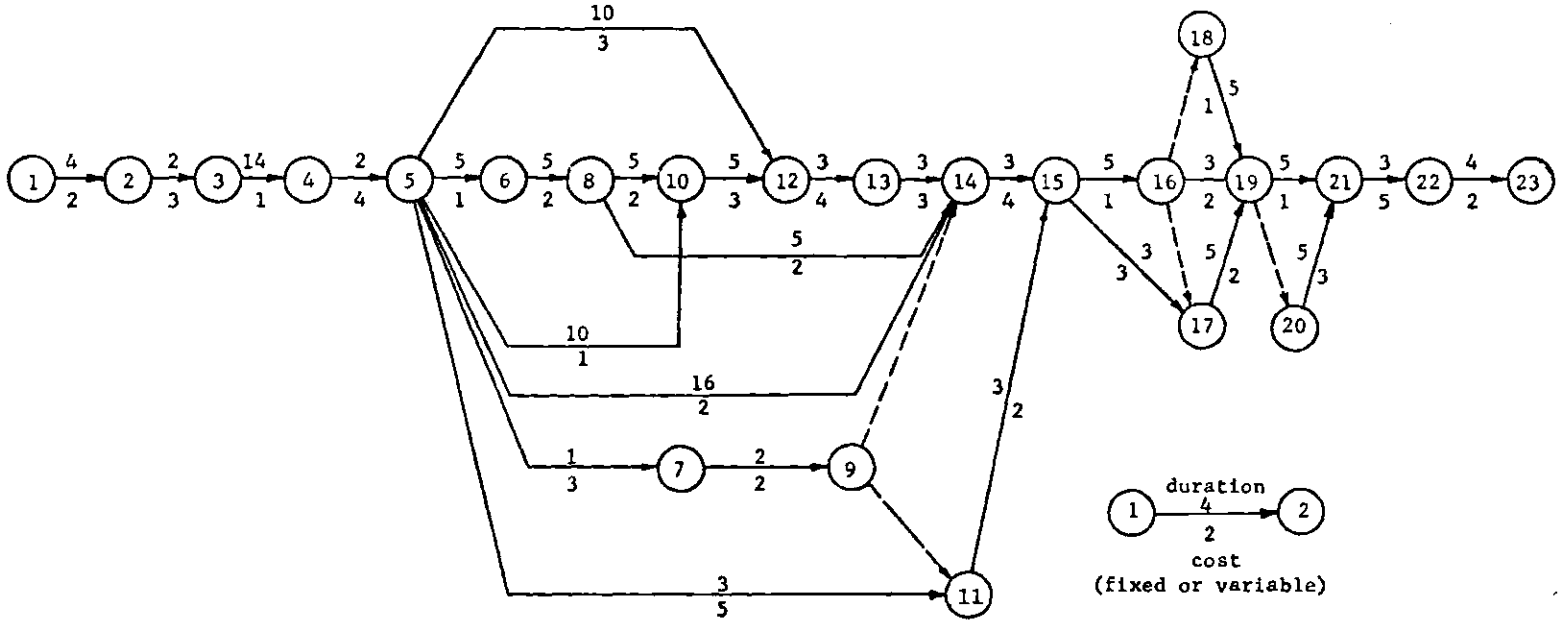


Figure 3. Network for One Cycle of a Five Cycle Project

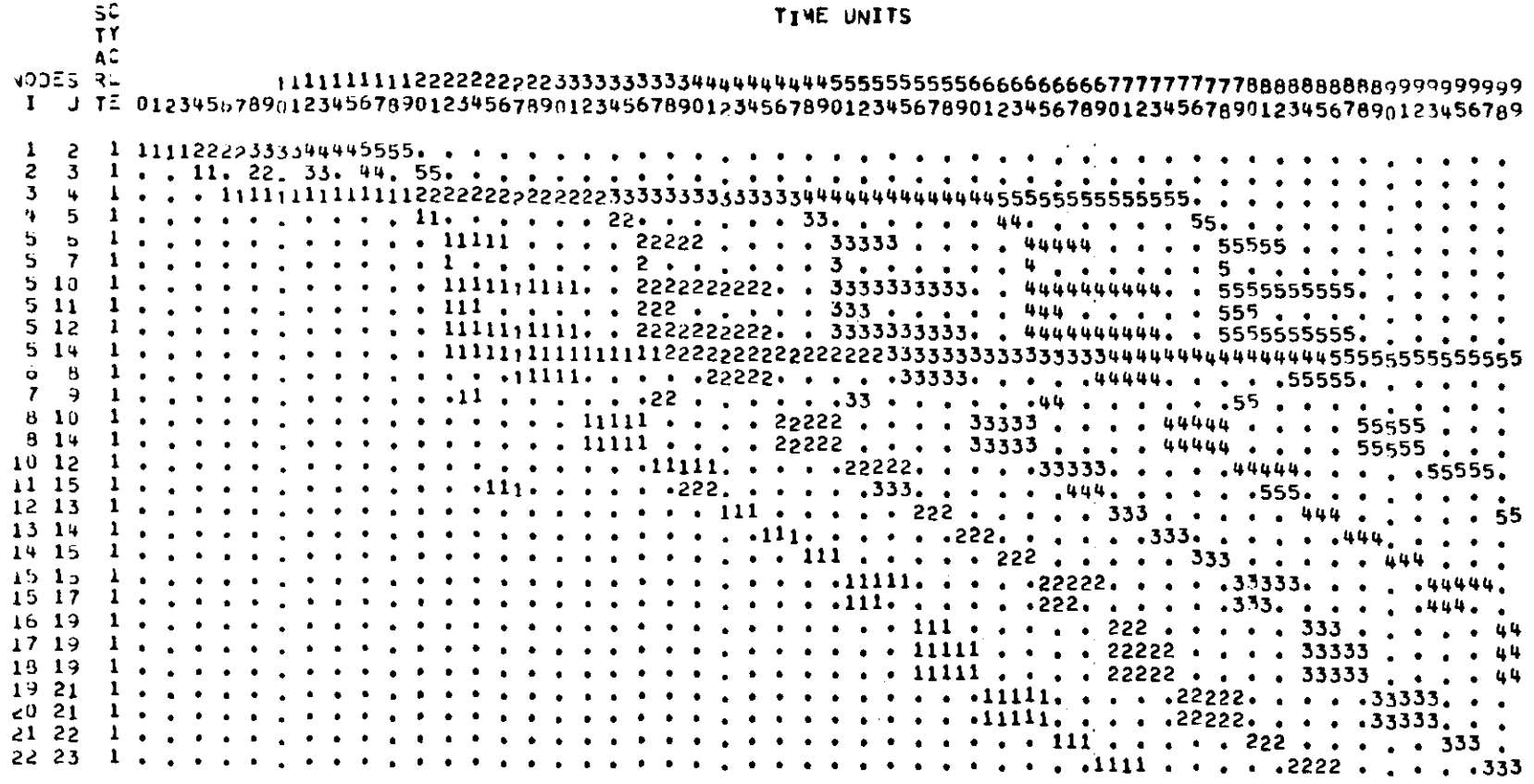


Figure 4. Gantt Chart of the Early Start Time Schedule for the Cyclic Project of Figure 3

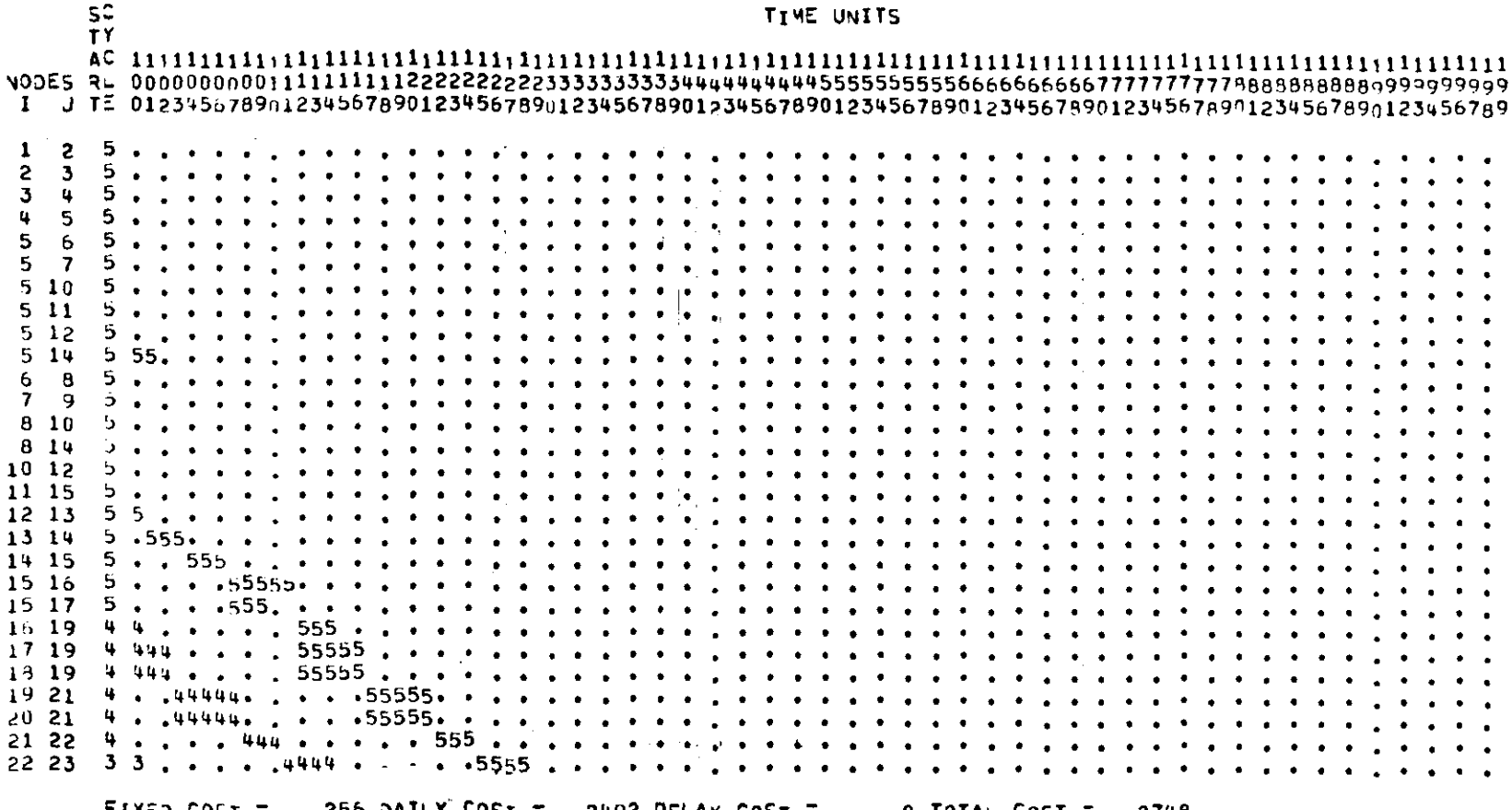


Figure 4. (Continued)

every activity is interrupted when going from one cycle to the next. However, an examination of the scheduled cycles will show that many interruptions could be eliminated or reduced in length by using some of the slack between an activity and its predecessors or successors. As an example, consider the schedule of activity (2,3). At its earliest start schedule there is an interruption between each cycle and the succeeding cycle. The schedule of the first cycle is fixed; note, however, that the start time of cycle two could be delayed to a time of 14, that cycle three could be delayed to a time of 16, and cycle four could be delayed to a time of 18, without violating any precedence relationships for start times of other activities. Then, there would be only one interruption--between cycles one and two--and all the other cycles would proceed from one to the next continuously.

The problem becomes more complex when looking at the entire project. Because of the interdependence of the activities in the project, it is not easy to determine how these interruptions can be removed from the most activities in order to get the "best" schedule. Utilizing available slack to make some activities continuous will usually force other activities to remain interrupted. If these interrupted activities are more important than the activity that is made continuous, it is quite obvious that the best solution will not be achieved. Hence, it would appear that if it is desirable to obtain some schedule that maximizes the positive effects of continuity of activities in the project, it would be necessary to solve the problems of: (1) ascertaining some measure of the relative importance of interrupting the continuity of each activity in the project; (2) determining how these measures may be used in an objective function; and

(3) developing some procedures that provide schedules which reduce the value of the objective function.

Purpose of the Research

It is the intended purpose of this research to advance the knowledge of cyclic project scheduling by attempting to provide answers to the problems identified above. As such, this research has been directed toward:

(1) definition of alternative penalty measures for activity interruptions in cyclic projects;

(2) development of objective functions for schedules of cyclic projects; and

(3) modification and extension of existing scheduling algorithms to reduce the value of different objective functions.

Assumptions Made

The following assumptions were made in this research:

(1) all projects considered are parallel multi-cycle projects;

(2) all references to continuity in projects will refer to close continuity;

(3) the cost or penalty incurred when an activity is interrupted is not dependent on the point in time when that interruption occurs;

(4) the completion time of the project may be delayed at some known cost or penalty;

(5) it is not possible to be working on more than one cycle of an activity at a particular point in time;

(6) once a cycle of an activity begins, it is carried out to its completion without interruption; and

(7) resource allocation and leveling will not be explicitly considered.

Survey of the Literature

Throughout the literature on network scheduling, there is little mention of cyclic projects. What research has been done on this topic, until recently, has not concerned to any extent the importance of continuity of activities.

Some of the better-known texts and references on network scheduling, such as those by Battersby (1), Wiest and Levy (8), and Woodgate (9), make no mention of cyclic projects at all. Burman (3) discusses cyclic projects, but only addresses the problem relating to the condensation of complex networks. He also defines continuity of activities but does not discuss it in detail and does not relate it to any problems in cyclic project scheduling. Moder and Phillips (7) also make very brief mention of cyclic projects but also only address the problem of condensation of complex networks.

An early paper on cyclic projects was presented by Burgess and Killebrew (2) but the problem they treat concerns minimization of the variance of activity level over the life of the project. Another paper, by Fisher and Nemhauser (6), is a valuable source for the definition of serial and parallel multi-cycle projects and an explanation of the simplified method of computing the activity early and late start times. However, the paper also does not treat the idea of continuity of activities; instead, a method is presented for obtaining the algebraic representation of a parallel multi-cycle project network based on the network for a single cycle.

The first significant research to discuss the problems of continuous scheduling of a cyclic project was that by Burney (4). Burney sought to maximize the continuity of a single activity in a cyclic project, which he called the "sub-contracted" activity. The methods he developed for accomplishing this were based on efficient allocation of resources. He also established the relationships between the sub-contracted activity and other important activities in the network, such as the longest activity in the network and the longest activity on the critical path of one cycle. His methods also enable the prediction of the minimum number of interruptions possible in the sub-contracted activity without having to actually schedule the project. It was his conclusion that these methods would be valuable in the planning of a project and in enabling a sub-contractor to make more efficient use of his resources and reduce the total costs of his portion of the project.

Following Burney's work, the most significant research to date has been that of Ferraz (5). Much of what will be developed in this research is a direct extension of the research which he undertook. Ferraz has developed a heuristic procedure that reduces the sum of the costs of activity interruptions in a cyclic project network. He assumes that there is a fixed cost associated with an interruption of each activity in the project and that the project must be completed at its earliest completion time. His procedure then determines the critical path over all cycles of the project and schedules the cycles of the activities that are a part of it. The procedure then puts each unscheduled activity into one of several different subsets of unscheduled activities. Different scheduling procedures are then applied to the activities in the various subsets. When all

activities have been scheduled, a low-cost schedule for the project has been obtained.

It appears that the procedure which Ferraz has developed in a good beginning from which to attack the complex problems of scheduling cyclic projects. Detailed study will show, however, that there is still much work remaining to be done. The assumptions which Ferraz has made and the results he achieved are somewhat narrow and incomplete and do not provide a thorough solution to the problem. The work presented here is an attempt to extend the work of Ferraz through the development of more comprehensive scheduling procedures which provide good solutions to the defined problems of continuously scheduling activities in parallel multi-cycle projects.

CHAPTER II

OBJECTIVE FUNCTIONS

In order to develop scheduling procedures for improving the continuity of cyclic projects there must be some means of defining the relative penalties for an interruption of activities in the project. In addition, there must also be an objective function which gives some quantitative indication of the significance of all interruptions in the schedule and a measure of effectiveness which shows how effective the scheduling procedure has been in reducing costs or other penalty measures.

As an example, consider the objective function offered by Ferraz (5). As an indicator of the importance of the interruptions in schedules of activities, the Ferraz procedure requires that a fixed cost be assigned to the occurrence of an interruption for each activity. The procedure then attempts to reduce the sum of the interruption costs of every activity, which is computed for all activities by summing the products of the number of interruptions in an activity times the interruption cost for that activity.

Though a very simple and easily applied objective function, the algorithm as presented by Ferraz is too narrow perspective from which to schedule cyclic projects. In many cases, there are other costs which may be applicable; or, it may not be possible to determine these costs at all. It could also occur that management had other considerations that were more important than costs alone and would be more interested in attaining other objectives. Therefore, it is necessary to consider other objective

functions and the relative factors that are necessary for their formulation.

Development of the Objective Functions

The objective functions which are presented here are based upon one or more of the following factors: (1) the number of interruptions in an activity; (2) the length of each interruption in an activity; (3) a fixed cost associated with each interruption in an activity; (4) a cost associated with the length of an interruption in the schedule of an activity; and (5) the delay of the completion of the project beyond its minimum completion time and the costs associated with the length of such a delay. Generally, these objective functions will reflect a measure of the efficiency of a schedule or the economics of a schedule. The efficiency measures will be based only upon the number of interruptions and their lengths, while the economic measures will reflect the costs resulting from various schedules.

Number of Interruptions

The first function considered is simply the sum of the number of interruptions in the schedules of all activities in the project. This would be computed by:

$$\sum_{i=1}^{M} \sum_{j=2}^{N} I(i,j) \qquad (1)$$

where

M = the number of activities,

N = the number of cycles in the project,

and the function I(i,j) is defined by

$$I(i,j) = 1, \text{ if } S_{i,j} - S_{i,j-1} - D_i > 0,$$
$$= 0, \text{ otherwise,}$$

and where

$S_{i,j}$ = the scheduled start of the i^{th} activity at its j^{th} cycle,

D_i = the duration of the i^{th} activity.

Though very easy to compute, this measure may give only a very crude indication of the value of a schedule. Though such a measure may indicate that it is possible to attain a very low number of interruptions in a schedule (indicating, perhaps, a very efficient schedule), it does not in any way account for the problems of: having many interruptions in some activities while having only a few interruptions in others; having interruptions of significant length; having high variability in the cost of interruptions among the activities of a project. Though it appears that this measure is easy to compute and may indicate to some extent the overall efficiency of the project schedule, it may not be of much help in the approximation of actual penalty costs.

Sum of Squares of the Number of Interruptions

Another objective function which could be based on the number of interruptions of activities in the project would compute the sum of squares of the number of interruptions in each activity of the project. Though not a very useful function, this would tend to alleviate one problem of the above function since it would tend to favor schedules which equalized the number of interruptions in activities. This function would be computed by:

$$\sum_{i=1}^{M} \left[\sum_{j=2}^{N} I(i,j) \right]^2 \qquad (2)$$

and where all the variables are as defined previously.

Total Interruption Time

Still another method would be to find the total amount of time in the schedule for the project that results from interruptions between cycles of activities. This function would be defined by:

$$\sum_{i=1}^{M} \sum_{j=2}^{N} \left(S_{i,j} - S_{i,j-1} - D_i \right) \tag{3}$$

where all variables are as defined previously.

Though a fairly easy function to calculate and understand, this function has obvious disadvantages. Though a crude measure that shows how much time there is in the project that is attributable to interruptions, it does not give any indication of the severity of the interruptions that remain in the schedule nor does it in any way consider the variance in the lengths of interruptions for different cycles of different activities.

Sum of Squares of the Lengths of Interruptions

Like the case for the number of interruptions in the schedule, perhaps a better function based on the lengths of the interruptions could be based on a sum of squares approach. This approach would tend to minimize the variance in the lengths of interruptions between cycles of activities, resulting in interruptions that would tend to be more the same duration for all activities. This would be computed as follows:

$$\sum_{i=1}^{M} \sum_{j=2}^{N} \left(S_{i,j} - S_{i,j-1} - D_i \right)^2 \tag{4}$$

where all variables are as defined previously.

Fixed Cost of Interruption

Though the four measures defined above are easy to define and use, their applicability would possibly be restricted to a few specialized cases. The failure of such objective functions is that they do not incorporate any measure of the relative importance of interrupting different activities; they treat the interruption of all activities equally and result in the minimization of the sum total of their effects on the entire project. Better functions can be formulated when it is possible to define costs associated with the interruption of specific activities because it is then possible to measure the effect that the interruption of any activity will have on the project. One of these functions, which has already been mentioned, is that which was used in the algorithm developed by Ferraz (5). This function is determined by a sum of the products, for each activity, of the activity's fixed cost of interruption times the number of interruptions in the schedule of all cycles of the activity. This would be given by:

$$\sum_{i=1}^{M} \sum_{j=2}^{N} \left(C_i\right) \left(I(i,j)\right) \tag{5}$$

where the function $I(i,j)$ is as defined previously and C_i is the fixed cost of interruption for the i^{th} activity.

This function is also relatively easy to calculate and certainly is a more useful one than any of the functions previously discussed. The cost calculated by this function could most likely represent the total amount of start-up and shut-down costs resulting from a schedule for a cyclic project and could be easily employed if this is the relevant cost

to be reduced. However, if other identifiable costs are considered, it will also be necessary to consider other economically based objective functions.

Variable (or Daily) Cost of Interruption

It might also be necessary to consider the costs that could occur from the length of an interruption of an activity. For example, some costs might be incurred due to idle manpower or idle equipment. In this case, it would be necessary to determine a per day cost that would be accrued for each day that the activity is interrupted. This could be called the variable cost of interruption and could result in an objective function of the form:

$$\sum_{i=1}^{M} (V_i) \left[\sum_{j=2}^{N} (S_{i,j} - S_{i,j-1} - D_i) \right] \tag{6}$$

where V_i is the variable cost of interruption for the i^{th} activity and the other variables are as defined previously.

Total Cost of Interruption

It is logical to assume that, in reality, it would most likely occur that both fixed and variable costs could result from the interruption of an activity. In this case, it would be necessary to consider a total cost of interruption that would be the sum of the fixed and variable costs. This would give an objective function of the form:

$$\sum_{i=1}^{M} \sum_{j=2}^{N} \left\{ \left[(C_i) \left(I(i,j) \right) \right] + \left[(V_i) \left(S_{i,j} - S_{i,j-1} - D_i \right) \right] \right\} \tag{7}$$

where all variables are as defined previously.

Cost of Delay

Previous research in cyclic project scheduling has not considered the idea of extending the project completion date in order to allow for more continuity of activities. This is obviously an important practical consideration. However, it must be recognized that any delay in the completion of the project may result in penalty costs. Thus the objective function should take into consideration the delay costs of a project which has a completion time extended beyond the minimum. The function is defined by:

$$\left(C_d\right)\left(T_{comp} - T_{min}\right) \tag{8}$$

where

C_d = the cost of each day's delay in completing the project,

T_{comp} = the actual completion time of the project,

T_{min} = the minimum completion time of the project.

Total Cost

Finally, it may be desirable to combine all of the above costs into one cost function which is a reflection of the costs incurred by activity interruptions and project delay. This function has the form:

$$\sum_{i=1}^{M}\sum_{j=2}^{N}\left\{\left[\left(C_i\right)\left(I(i,j)\right)\right] + \left[\left(V_i\right)\left(S_{i,j} - S_{i,j-1} - D_i\right)\right]\right\} + \left(C_d\right)\left(T_{comp} - T_{min}\right) \tag{9}$$

where all variables are as defined previously.

Upper and Lower Bounds on Objective Functions

The objective functions described above provide a base from which one may specify the objectives to be accomplished by a scheduling procedure for cyclic projects. However, some care must be used in interpreting the final value that may be yielded by the objective function in the scheduling procedure. If the initial value of the objective function is taken to be the value determined from the early start schedule of all activities, it may occur that the final value of the objective function at the completion of the scheduling procedure will not differ significantly from the initial value. However, this will not necessarily mean that a good solution has not been obtained. It is very important to recognize that there are upper and lower bounds on the values of the different objective functions.

Ferraz presented some ideas on what the upper and lower bounds for the fixed cost objective function could be. These same ideas may be extended to cover the additional objective functions which have been described. However, it should be pointed out that the bounds which Ferraz described were developed for the case of scheduling at the minimum completion time and are not applicable for the case of extending the project completion date.

Since the objective of the scheduling procedure will always be to minimize one of the objective functions, the lower bound on the function will be of greater importance than the value of the upper bound. Ferraz approached the problem of determining the lower bound by considering the scheduling of each activity for maximum continuity independently of the schedules of other activities in the project (i.e., ignoring the fact that the schedules may be infeasible). Once the early and late start times

for every activity have been determined, a heuristic procedure is applied to each activity. The last cycle of the activity is scheduled at its early start time. Then, proceeding from the next-to-last cycle to the first, the following rule is applied: if a cycle of the activity cannot be made continuous with its succeeding cycle (i.e., the scheduled start of the cycle would not be within the bounds determined by that cycle's early and late start times), then the cycle is scheduled at its early start time. When the schedule for each activity has been determined, the objective function is evaluated to give the lower bound. (The heuristic procedure referred to above is called the "backward algorithm" and will be more fully discussed in Chapter III.)

The lower bound may always be computed, even for the case where the project completion date has been extended. When computing the lower bound for an alternate completion date, the lower bound of the costs or penalties resulting from the interruption of activities is computed exactly as described above; the later completion time will have no effect on the use of the heuristic procedure which is applied, independently, to each activity. However, added to the resulting lower bound on the interruption cost or penalty will be the cost of delay. This will be a value computed by the product of the difference in completion times and the per day delay cost, as given by function (5).

Ferraz also discussed the determination of an upper bound on the value of the fixed cost objective function which he employed. However, since the objective of these scheduling procedures is cost or penalty minimization, the use of an upper bound is relatively unimportant. The

upper bounds on the objective functions may also be rather hard to determine. For the case where the project is scheduled for minimum completion, there are several ways to determine what this upper bound should be. One way is to consider the number of cycles minus one as the maximum number of interruptions for each activity. This would give an upper bound on the number of interruptions for all activities in the project. However, this upper bound would be too high; some activities will never have any interruptions in a minimum completion time schedule. A better approach is to consider the schedules of the activities at their early start times. This is the schedule that would normally be used and is the one at which the scheduling procedure would begin. It would then present a better upper bound for the appropriate objective function.

When the project has its completion time extended, the appropriate upper bound is much harder to determine. With more time to complete the project, there is a possibility that some activities could have more interruptions or interruptions of greater length. However, no procedure has been developed which can give the greatest possible cost for a given completion time. Thus, the upper bound would not be of much value when scheduling under such a constraint.

Ferraz developed his ideas on lower and upper bounds in order to develop a measure of effectiveness for the cyclic project scheduling procedure. However, it is applicable only in the case of minimum completion time scheduling. Since the assumption here is that the completion time may be extended, it will not be possible to determine a similar measure of effectiveness for the cyclic project scheduling procedures to be developed in this research. Since it will always be possible to compute a

lower bound for any objective function, it is suggested that one indication of the effectiveness of a schedule will be the difference in the cost yielded by the algorithm as compared to the lower bound on the cost of the project for a given completion time.

Now that the objective functions have been developed, they will be employed in a procedure to reduce their respective values. The development of such a procedure is considered in the next chapter.

CHAPTER III

DEVELOPMENT OF THE SCHEDULING ALGORITHM

The algorithm which Ferraz has developed for the scheduling of multi-cycle projects is a heuristic procedure which attempts only to reduce the fixed cost of interruption of activities in the project (5). While the procedure represents a significantly advanced technique for cyclic project scheduling, it appears that there is need for simplification, clarification, and improvement of certain parts of the algorithm. Also, with the additional objective functions defined in the previous chapter, it will be advantageous to incorporate the additional objective functions into the scheduling algorithm. This chapter will focus on a discussion of a modified scheduling algorithm.

The Scheduling Algorithm

The scheduling algorithm presented in this chapter is basically an adaptation of the procedure first offered by Ferraz (5). The algorithm offered here includes several key improvements over the Ferraz procedure. For any steps where changes have been made, the approach originally used by Ferraz will be pointed out and an explanation of the modification will be given.

Before actually discussing the algorithm, it should be noted that only the network representing a single cycle of the project will be required. However, care should be taken in the construction of the network.

Like many other scheduling algorithms, this procedure requires an ordered numbering of the nodes of the diagram. If G(N,A) is the network representing the project, where N is the set of nodes, and the set of activities, A, is given by

$$A = \{(i,j) | (i) \ll (j);\ i,j \in N\} ,$$

then it is necessary that $j > i$ for all activities (i,j). This requirement is necessary for the proper and efficient use of Ferraz's algorithm.

The first step of the procedure is to arrange the activities in the network in a specific order. This order is determined by the sum of each activity's i-node plus its j-node. The activities are arranged according to the increasing values of their sums. As an example, the ordered set for the network of Figure 3 would begin with activity (1,2), then activity (2,3), then activity (3,4), etc. This ordered set will be denoted by R and will represent the set of unscheduled activities in the project.

When the ordered set R has been formed, the early and late start schedules for all cycles of each activity may be computed. These schedules are determined using the normal CPM techniques, except the specialized structure and precedence relationships of parallel multi-cycle projects are capitalized upon in order to allow more efficient scheduling. The mathematical statements of the scheduling are presented below.

To compute the early start schedule:

$$ESS(R_m,1) = \max_{k\epsilon\alpha(R_m)} \left[ESS(R_k,1) + D_{R_k} \right] , \tag{10}$$

$$ESS(R_m,n) = \max\left\{ \left[ESS(R_m,n-1) + D_{R_m} \right] , \max_{k\epsilon\alpha(R_m)} \left[ESS(R_k,n) + D_{R_k} \right] \right\} , \tag{11}$$

where

$m = 1,2,\ldots,M$, the number of activities in the project,

$n = 2,3,\ldots,N$, the number of cycles in the project,

R_i = the i^{th} activity in the set R,

$ESS(R_i,j)$ = the early start time of the j^{th} cycle of activity R_i,

D_{R_i} = the duration of activity R_i,

$\alpha(R_i)$ = the set of activities which are direct predecessors of activity R_i.

When the early start schedule has been obtained, the minimum completion time of the project has also been determined. This is given by the completion time of the last cycle of the last activity in the set R:

$$T_{min} = ESS(R_M,N) \tag{12}$$

The late start schedule is then determined as follows:

$$LSS(R_M,N) = T_{min} - D_{R_M} \tag{13}$$

$$LSS(R_m,N) = \min_{k\epsilon\beta(R_m)} \left[LSS(R_k,N) - D_{R_k} \right] , \tag{14}$$

$$LSS(R_m,n) = \min\left\{ \left[LSS(R_m,n+1) - D_{R_m} \right] , \min_{k\epsilon\beta(R_m)} \left[LSS(R_k,n) - D_{R_k} \right] \right\} , \tag{15}$$

where

$m = M, M-1, \ldots, 2, 1,$

$n = N-1, N-2, \ldots, 2, 1,$

$LSS(R_i,j)$ = the late start time of the j^{th} cycle of activity R_i,

$\beta(R_i)$ = the set of activities which are direct successors of activity R_i.

The other variables are the same as those defined for the computation of the early start schedule.

When the computation of the early and late start time schedules has been completed, an activity in the project may be scheduled. This activity will be the activity in the project which has its early start schedule equal to its late start schedule in every cycle. As Ferraz has demonstrated, there will always be at least one such activity in the project and it will be either the activity with the longest duration of all activities in the project or the activity on the critical path of one cycle of the project which has the longest duration of any activity on the critical path. The selection of one of these two activities will depend on the number of cycles in the project. The activity will have its schedule continuous over all cycles. Because such an activity has its schedule fixed for all cycles, it may be scheduled and removed from the set R.

This activity which is first scheduled will be called the "pacing activity" because its length is the most significant in the project and therefore sets its "pace." It should be noted that because its early and late start times are equal in every cycle, all of its cycles lie on the critical path of the project. Therefore, it is also the most critical

activity in the project, because if any cycle of the activity is delayed, the project must be delayed as well.

Ferraz offered a quite different procedure for determining the scheduling of the pacing activity. His "method for determining the critical path" is based on the computation of an index which is derived from the length of the longest activity in the project and the length of the longest activity on the critical path of one cycle of the project. When compared with the number of cycles in the project, the pacing activity may be determined. Ferraz's procedure is inefficient computationally, since the pacing activity can be easily determined from the early and late start schedules.

With the pacing activity scheduled, the remaining unscheduled activities are considered. Certain activities in the set R are decomposed into one of three subsets which are defined as follows:

Subset L--a subset of R formed by all activities which have all of their predecessors completely scheduled, but not all successors;

Subset M--a subset of R formed by all activities which have all of their successors completely scheduled, but not all predecessors;

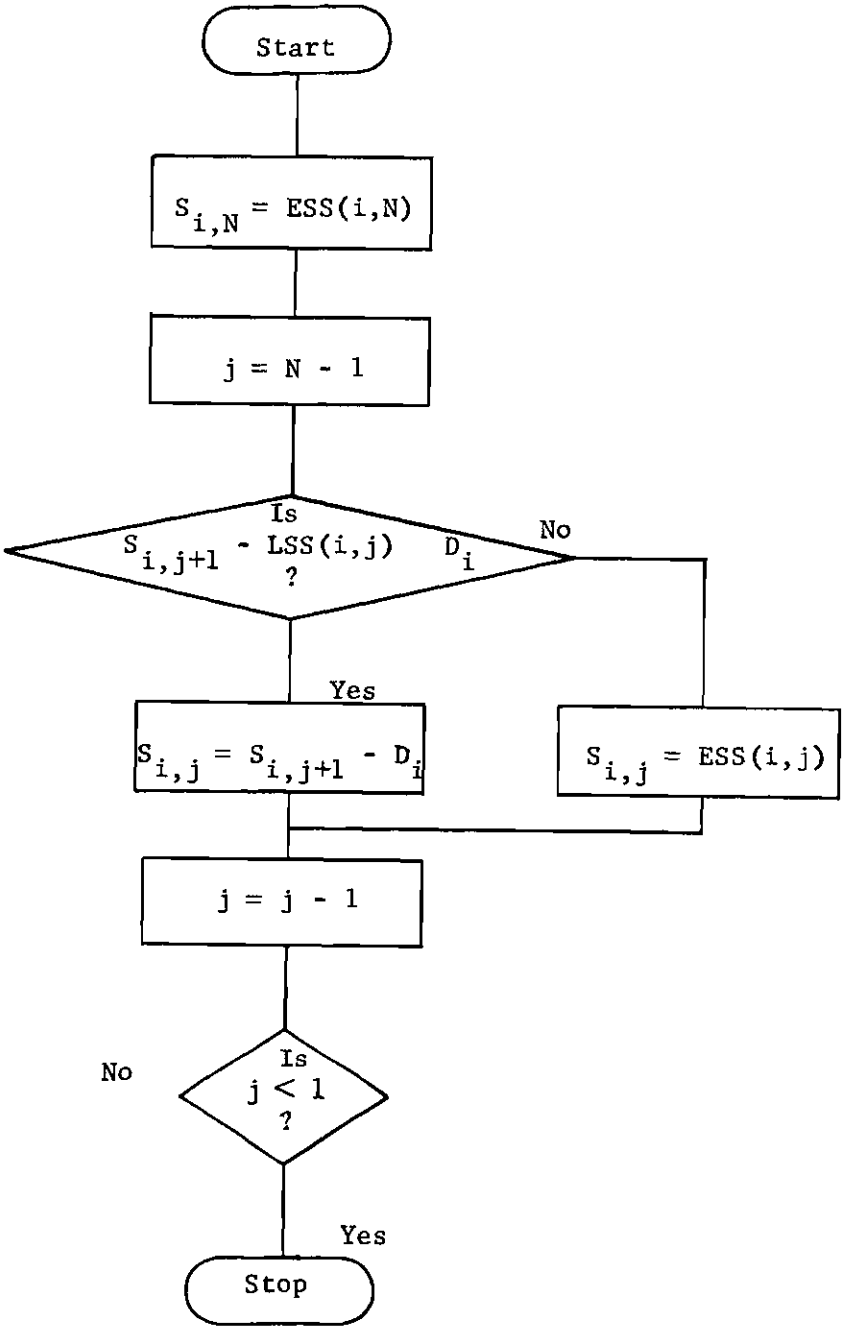
Subset K--a subset of R formed by all activities which have all of their predecessors and all of their successors completely scheduled.

An attempt is now made to schedule each activity in the sets K and L using the "backward algorithm" and each activity in the set M using the "forward algorithm." The backward and forward algorithms are heuristic procedures which seek to establish close continuity between consecutive cycles of an activity. The algorithms take their names from the direction, with respect to the cycles of the activity, that each attempts to schedule

an activity: the forward algorithm from the first cycle to the last cycle and the backward algorithm from the last cycle to the first cycle.

The backward algorithm is the first method used by the scheduling procedure. The schedule which it generates will be called the BS schedule for an activity. To get the BS schedule, the activity is scheduled in its last cycle at its early start time. Then, the next-to-the-last cycle is checked to see if it may be scheduled continuously with the last cycle without increasing the project duration (i.e., the start time of the next-to-the-last cycle, if continuous with the last cycle, would not be greater than the late start time for that cycle). If this condition is satisfied, then an attempt is made to schedule the next cycle continuous with the next-to-last cycle. If the condition is not satisfied, the next-to-the-last cycle is also scheduled at its early start time and an attempt is made to make the next cycle continuous with it. This procedure is repeated until the first cycle has been scheduled. A flow chart depicting the procedure is presented in Figure 5.

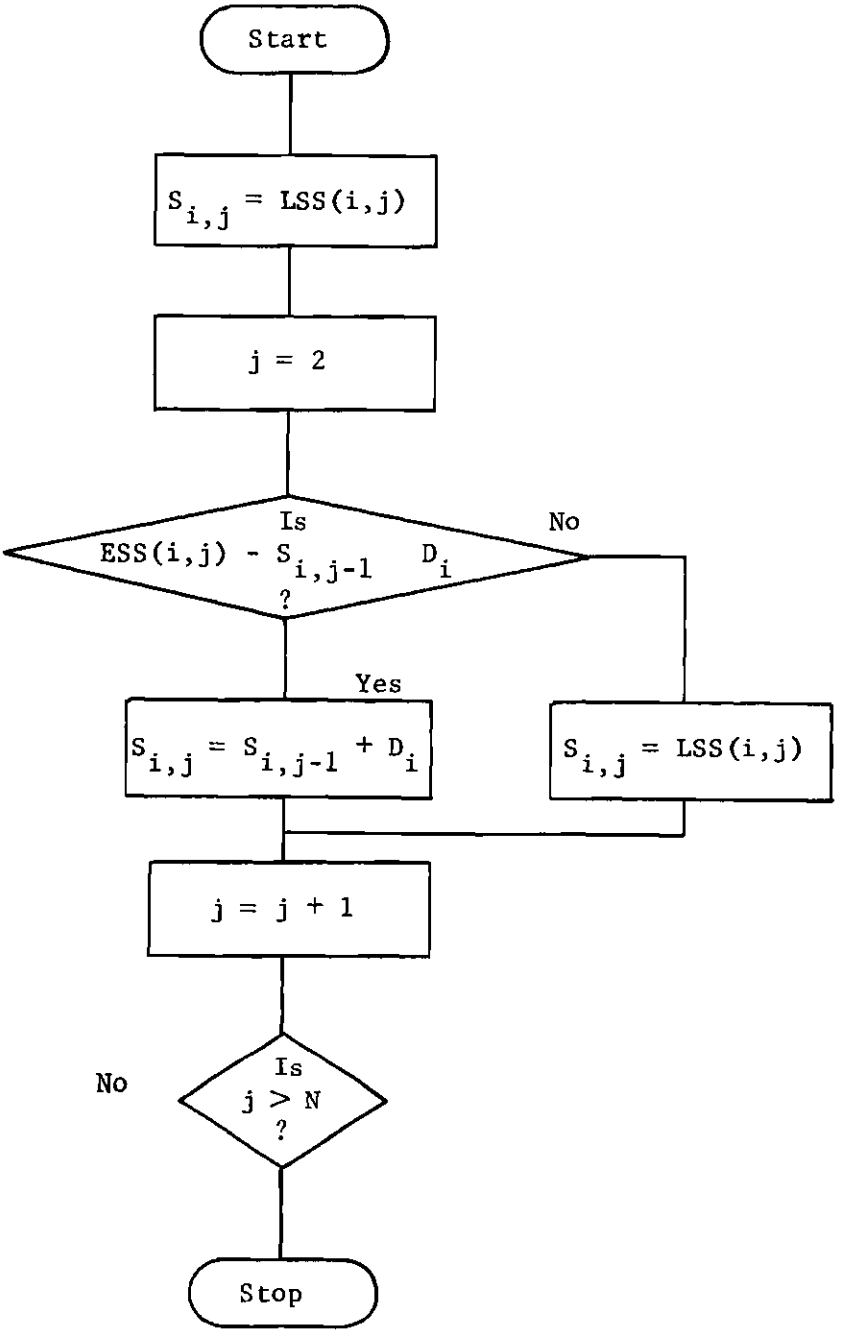
The forward algorithm is similar to the backward algorithm, except that the procedural steps are reversed. The first cycle of the activity is scheduled at its late start time. The second cycle is then checked to see if it may be feasibly made continuous with the first cycle (i.e., the schedule for the second cycle would not be less than the early start time for the cycle). If this condition is met, the second cycle is scheduled and the third cycle is considered. If the condition is not satisfied, the second cycle is scheduled at its late start time and the procedure starts over with cycle 3. The procedure continues until the last cycle is scheduled. The schedule that results is called the FS schedule for the activity. A flowchart of the algorithm is given in Figure 6.



Notation:

N = the number of cycles in the project
$S_{i,j}$ = the scheduled start time of cycle j of activity i
ESS(i,j) = the early start time of cycle j of activity i
LSS(i,j) = the late start time of cycle j of activity i
D_i = the duration of activity i

Figure 5. Flowchart of the Backward Algorithm



Notation:

N = the number of cycles in the project
$S_{i,j}$ = the scheduled start time of cycle j of activity i
ESS(i,j) = the early start time of cycle j of activity i
LSS(i,j) = the late start time of cycle j of activity i
D_i = the duration of activity i

Figure 6. Flowchart of the Forward Algorithm

In applying these algorithms to the unscheduled activities in R, the first subset of unscheduled activities considered are those in the set L. The first activity in the ordered set R that is in L is selected and the backward algorithm is applied. The resulting BS schedule is then checked for feasibility with the early start schedules of all successors of the activity which is being considered. If feasibility holds in all cases, the activity is considered scheduled, the activity is deleted from the set R, the subsets are updated, and the next activity in the ordered subset L is considered.

If the activity does not meet the feasibility criterion with respect to the early start schedules of its successors, it cannot yet be scheduled. Instead, the activity is placed in a subset of L, called L', and will remain there until considered at a later stage of the procedure. The next activity in L is then considered; or, if L is empty, then the procedure goes to the next step, scheduling activities in the set M.

The scheduling of activities in the set M is exactly opposite to that of the activities in the set L. The activities in R that are in the set M are considered in reverse order; that is, the last activity in R that is in M is selected first, then the next-to-last, etc. The forward algorithm is applied to the activity under consideration. The resulting FS schedule is then checked for feasibility with respect to the late start schedules of all predecessors of the activity. If feasibility holds in all cases, the activity is scheduled, it is deleted from the set R, the subsets are updated, and the next activity in the ordered subset M is considered.

Like activities in set L, if an activity in M does not meet the

feasibility criterion with respect to its FS schedule, the activity is added to a subset called M', to be considered at a later stage of the procedure along with the activities in L'. The next activity in M is then considered, or, if subset M is empty, the activities in set K are considered.

For the activities in set K, all activities that precede or succeed these activities have been completely scheduled. Therefore, simple application of the backward algorithm (or forward algorithm) will result in a feasible schedule. When all activities in K have been scheduled, it is necessary to determine if any activities remain unscheduled. If the set R is empty, then the scheduling procedure is finished.

If the set R is not empty, it is an indication that not only are there activities remaining that are unscheduled, but activities remain in the subsets L' and M' which resulted from not being able to schedule activities in the sets L and M earlier. The scheduling of these activities is accomplished using a heuristic bounding procedure which Ferraz called the "computation of ICL and ICM."

ICL is a value representing the lower bound on the incremental increase in the fixed costs of interruption if all the activities in the set L' are scheduled according to their BS schedules. This is computed as follows:

1. Find the BS schedule for each activity that is in R but not in L'.

2. Using the fixed interruption cost for each activity and the fixed interruption cost objective function, compute the fixed interruption cost for the activities that remain in R but which are not in L'.

3. Find the BS schedule for each activity in L'.

4. Based on the BS schedules for the activities in L', change the early start schedules of the activities in R which are not in L' in order to maintain feasibility of the schedules.

5. Using the updated early start times, recompute the BS schedules of the activities in R that are not in L'.

6. Recompute the fixed cost of interruption for the activities in R which are not in L'.

7. Equate ICL to the difference between the cost obtained in step (6) and the cost obtained in step (2).

Similarly, ICM is the lower bound on the incremental increase in the fixed cost of interruption if all the activities in the set M' are scheduled to their FS schedules. This is found by a procedure analogous to that of computing ICL:

1. Find the BS schedule for each activity that is in R but not in M'.

2. Compute the fixed interruption cost for each activity that is in R but not in M'.

3. Find the FS schedule for each activity that is in M'.

4. Based on the FS schedules for the activities in M', change the late start times of the activities in R which are not in M' in order to maintain feasibility of the schedules.

5. Using the updated late start times, recompute the BS schedules of the activities in R that are not in M'.

6. Recompute the fixed cost of interruption for the activities in R which are not in M'.

7. Equate ICM to the difference between the costs obtained in step (6) and step (2).

The values of ICL and ICM have now been obtained. In order to obtain the best schedule based on these lower bounds, the branch with the least lower bound is selected. Depending on the lower bound which is selected, different procedures will result.

If ICL is selected as being lower, the following will occur:

A. All activities in L' are scheduled to their BS schedules, the activities are removed from the set R, and the subsets are updated.

B. All activities remaining in R have their early start schedules updated to maintain feasibility with the activities being scheduled out of L'.

C. The scheduling procedure returns to the step of scheduling the first activity in the set L.

If ICM is lower, a similar series of steps are carried out:

A. All activities in the set M' are scheduled to their FS schedules, the activities are removed from the set R, and the subsets are updated.

B. All activities remaining in R have their late start schedules updated to maintain feasibility with the activities being scheduled out of set M'.

C. The scheduling procedure returns to the step of scheduling the last activity in the set M.

This description represents, in detail, the scheduling algorithm as developed by Ferraz, with one exception. In addition to the change in the method for determining the pacing activity, two steps were eliminated entirely from the procedure because they were found to be unnecessary. Ferraz, in determining the feasibility of BS schedules for activities in set L, chose to place activities that did not meet the feasibility criterion

into two different subsets. These subsets represented the activities in the set L whose BS schedules were not equal to their own early start schedules (which he called set L') and whose BS schedules were not feasible with the early start schedule of at least one successor (which he called set L''). It should be obvious that whether or not an activity is equal to its own early start schedule is relatively unimportant; there is concern only when an activity's BS schedule is not feasible with a successor's early start schedule. Ferraz duplicates the effort of scheduling activities in L by considering two separate subsets, L' and L'', when only one is needed. As a result, a scheduling step in his procedure was eliminated (checking for feasibility with the activity's own early start schedule) and another subset (L'') was eliminated. In the modified procedure just described, the set L' represents Ferraz's set L''; the set L' as described by Ferraz is eliminated. Because of an analogous situation regarding activities in the set M, a total of two scheduling steps in the procedure and two additional subsets were eliminated.

This discussion concludes the description of the modified scheduling procedure for reducing the fixed interruption costs of cyclic projects. Briefly, the procedure may be summarized as follows. The early and late start times are first computed and then used to obtain schedules of activities so that they may be employed by algorithms which reduce the number of interruptions in each activity. The procedure works from the "front" of the project network, left-shifting activities in time in order to reduce interruptions. When it is no longer possible to schedule these activities, the procedure goes to the "back" of the network and attempts to right-shift activities to reduce interruptions. When necessary, a

procedure is applied to make decisions concerning activities that could not be immediately scheduled out of their respective sets. As the procedure works to the "middle" of the project network, the left-shifting and right-shifting of the scheduled activities at the "front" and "back" of the network preserves slack that may be used in reducing the interruptions in remaining unscheduled activities. When all activities have been scheduled, the procedure is completed and a low-cost schedule has been obtained.

As an example, the procedure was applied to the problem that was presented in Figures 3 and 4. The resulting schedule is shown in Figure 7. A detailed presentation of the use of Ferraz's procedure as applied to other sample problems can be found in Ferraz (5).

Modifications and Improvements of the Procedure

Though the procedure of Ferraz was modified somewhat to make it more efficient, there are two primary improvements which can be used to extend considerably the usefulness of the algorithm. The two improvements involve the use of additional objective functions and the reduction of activity span.

Employing Additional Objective Functions

In developing his algorithm, Ferraz defined only a fixed cost of interruption for each activity and the use of a fixed cost objective function (5). For a more comprehensive scheduling procedure, it is important that the other objective functions also be considered.

The modified scheduling procedure may be adapted to the use of any of the objective functions described in Chapter II. The objective function should be selected before beginning the scheduling procedure. When it is

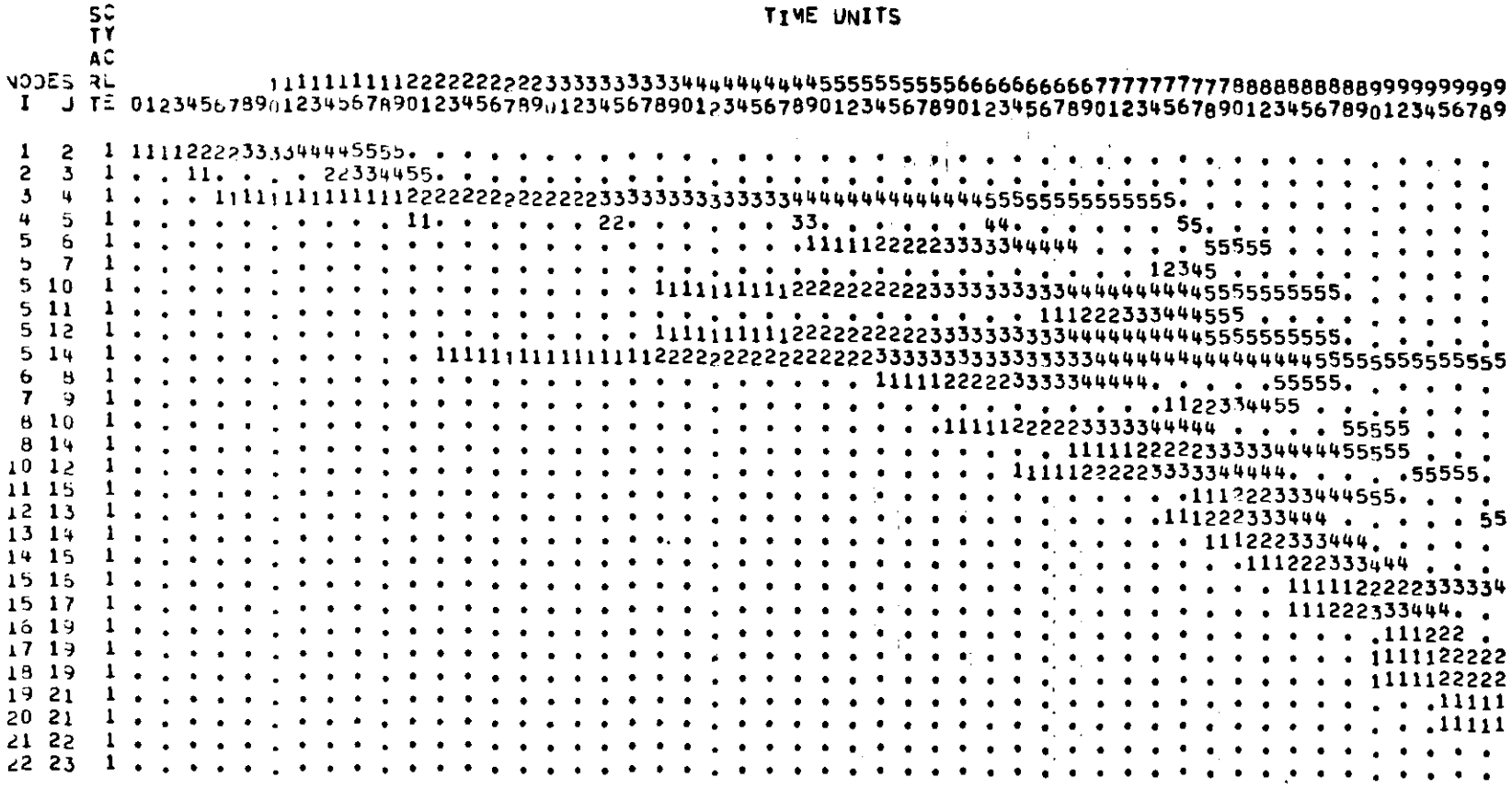


Figure 7. A Schedule for the Five Cycle Project of Figure 3

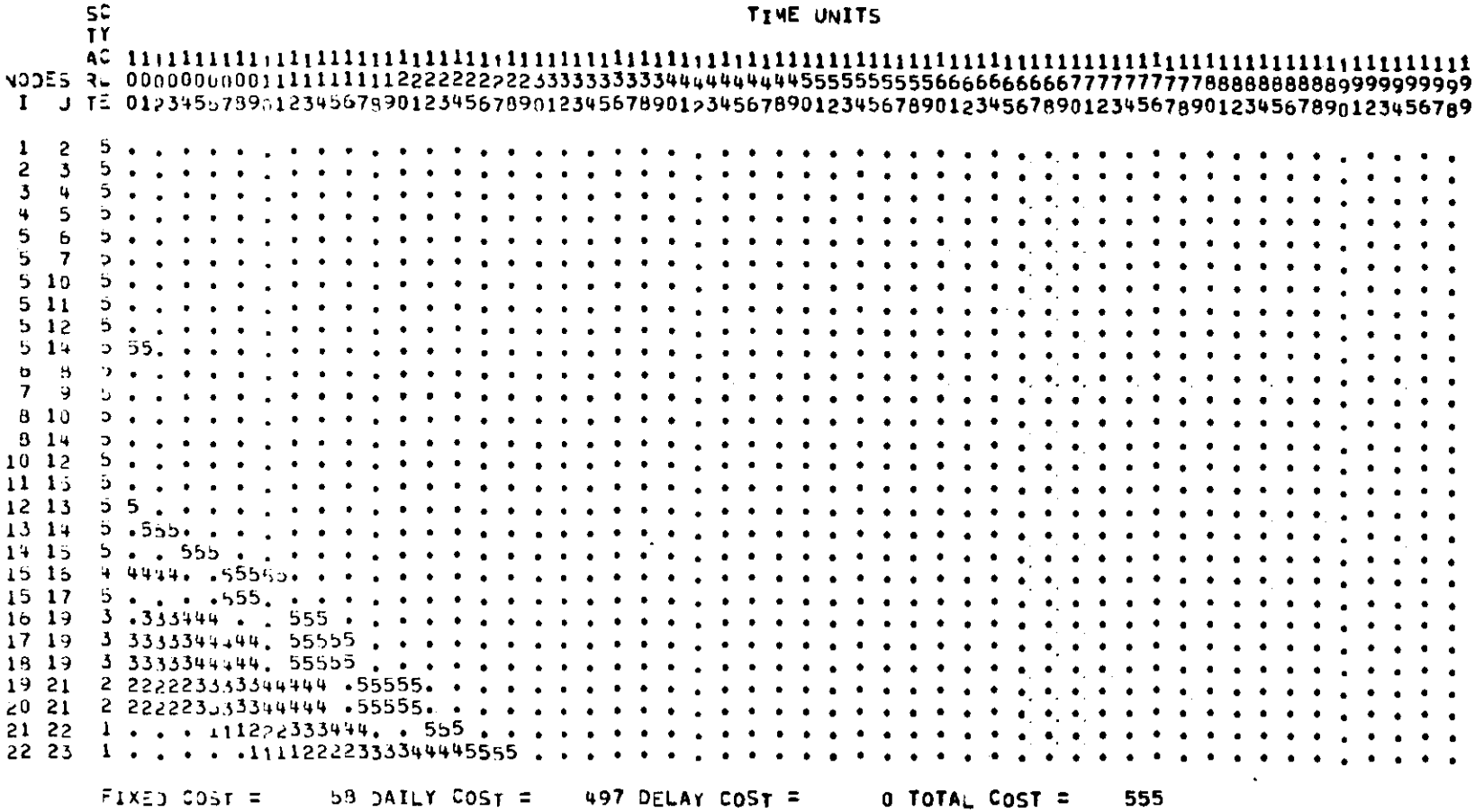


Figure 7. (Continued)

necessary to compute the values of ICL and ICM, the appropriate objective function may be used to compute the lower bound. The objective functions described by Equations (1), (2), and (5) may be employed directly without any additional modifications to the algorithm. However, the functions which are dependent on the lengths of interruptions will require an additional modification: shifting to reduce activity span.

Reducing the Span of Activities

One problem that Ferraz did not consider in the development of the procedure was the resulting lengths of the interruptions that did occur. In order to develop the ideas regarding this, the term "activity span" will be defined.

The span of an activity will be defined as the time elapsed from the beginning of the first cycle of an activity to the completion of the last cycle of the activity. Mathematically, this is given by

$$SPAN_i = S_{i,N} + D_i - S_{i,1} , \tag{16}$$

or

$$SPAN_i = \Big((N)(D_i) \Big) + \sum_{j=2}^{N} \Big(S_{i,j} - S_{i,j-1} - D_i \Big) \tag{17}$$

where $SPAN_i$ is the span of the i^{th} activity and the other variables are as defined earlier.

The problem that occurs in the scheduling procedure is that while it reduces the number of interruptions in the project schedule, it is not

yet a span reducing algorithm. There is currently no procedure in the algorithm to insure that interruptions will be of near minimal length. As an example, consider the schedule of activity (16,19) in Figure 7. Note the difference between the completion times of cycles one and two of this activity and the start times of cycles one and two of its only successor, activity (19,21). There is one unit of slack between the completion of cycle one of activity (16,19) and the beginning of cycle one of activity (19,21). Without destroying the feasibility of the schedules of activities (16,19) and (19,21), it would be possible to shift the start time of cycles one and two of activity (16,19) and thus decrease the span of the activity by one unit. Additionally, it can be seen that it would also be possible to right shift cycles three and four of activity (16,19) by one unit.

Though the shifting of cycles three and four has no effect on the span of activity (16,19) (only the shifting of cycle one will), their shift will have a significant effect on the schedules of other activities. By the shifting of cycles three and four of activity (16,19), there is one unit of slack between cycle four of activity (15,16) and cycle four of activity (16,19); therefore, activity (15,16) may be right-shifted and have its span reduced by one unit. In turn, activity (14,15) then may be shifted, then activity (13,14), etc.

In effect, the shifting of activity (16,19) starts a chain reaction, opening up the possibility of shifting more activities. Table 1 gives a summary of the shifting and span reduction that can be performed on the schedule of Figure 7. It should be noted that these span reductions are

Table 1. Span Reduction of Activities for the Schedule of Figure 7

Activity	Length of Reduction	Cost/Unit (V_i)	Cost Reduction
4 - 5	2	4	8
5 - 6	1	1	1
6 - 8	1	2	2
8 - 10	1	2	2
10 - 12	1	3	3
12 - 13	1	4	4
13 - 14	1	3	3
14 - 15	1	4	4
15 - 16	1	1	1
15 - 17	3	3	9
16 - 19	1	2	2

Total Cost Reduction = 39

quite significant. If the cost figures associated with the network of Figure 3 are assumed to be variable costs of interruption, the shifting of the schedule in Figure 7 results in a reduction from a variable interruption cost of 497 yielded by the scheduling procedure to a variable interruption cost of 458. Figure 8 is a Gantt chart of the schedule of Figure 7 after shifting and span reduction.

The reason for the unnecessarily lengthy interruptions yielded by the procedure is easily understood by considering the schedules produced by the backward algorithm. For example, in the backward algorithm, the first cycle is scheduled last. If the first cycle cannot be made continuous with the second cycle, it is scheduled at its early start time. The problem with this is that this may be too early to schedule the first cycle. But the backward algorithm is applied only to those activities which have all of their predecessors scheduled, and not their successors (set L). If the schedule of the first cycle of all successors were known, it is possible that a later start time for cycle one (or any other cycle which could not be scheduled continuously with the succeeding cycle) could be determined. But, since the schedules of successors are unknown, there is no choice but to schedule the cycle as early as possible in order to guarantee feasibility with the schedules of successor activities. This is exactly what happened to activity (16,19) and some of its predecessors in the computation of the schedule of Figure 7.

Similarly, the use of the forward algorithm may result in some activities having some cycles beginning too late. Therefore, it may be necessary to left-shift the cycles of some activities that were scheduled out of the set M. There were no such occurrences in the schedule of

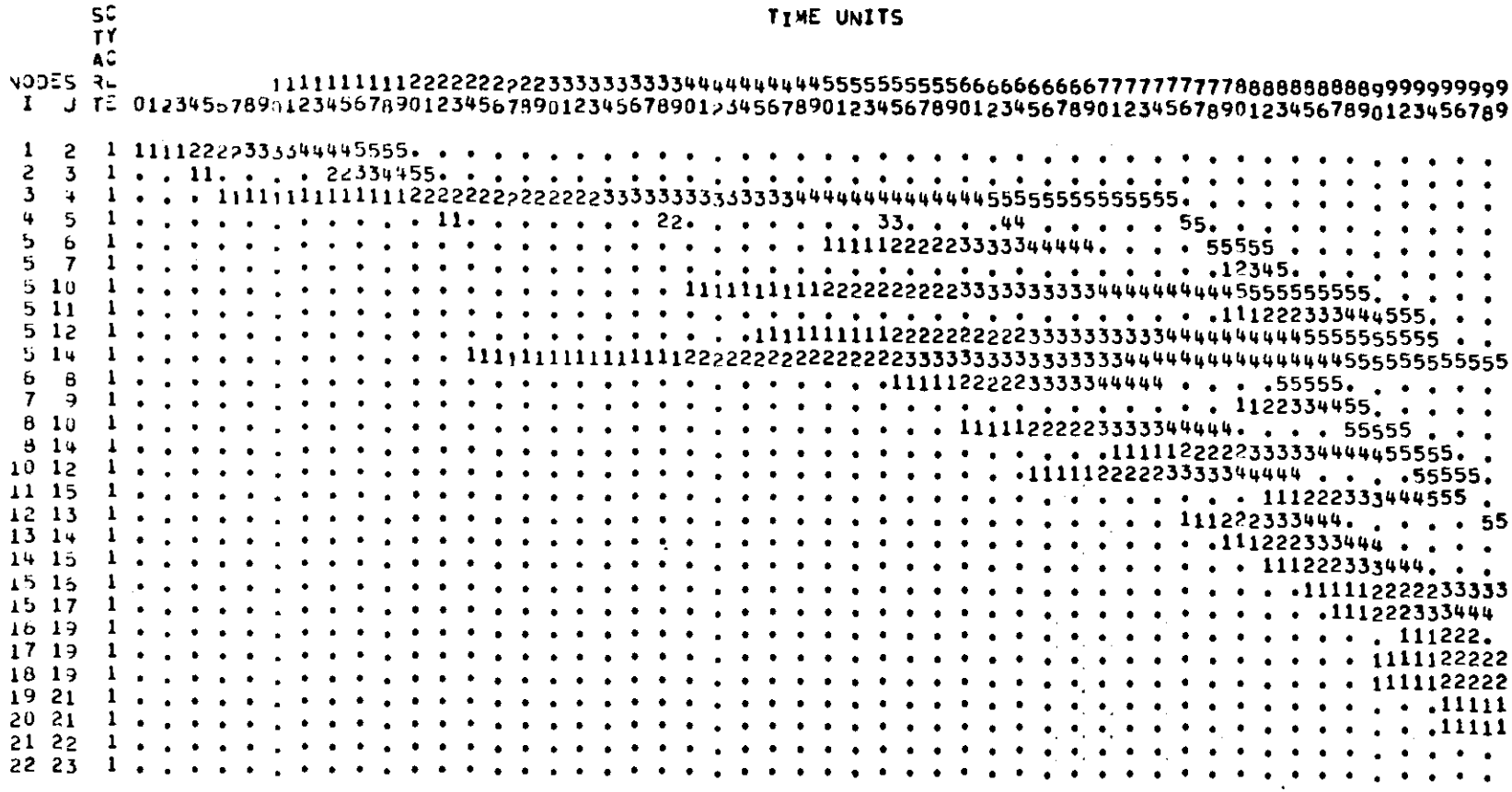


Figure 8. Five Cycle Project Schedule After Reducing the Span of Activities in the Schedule of Figure 7

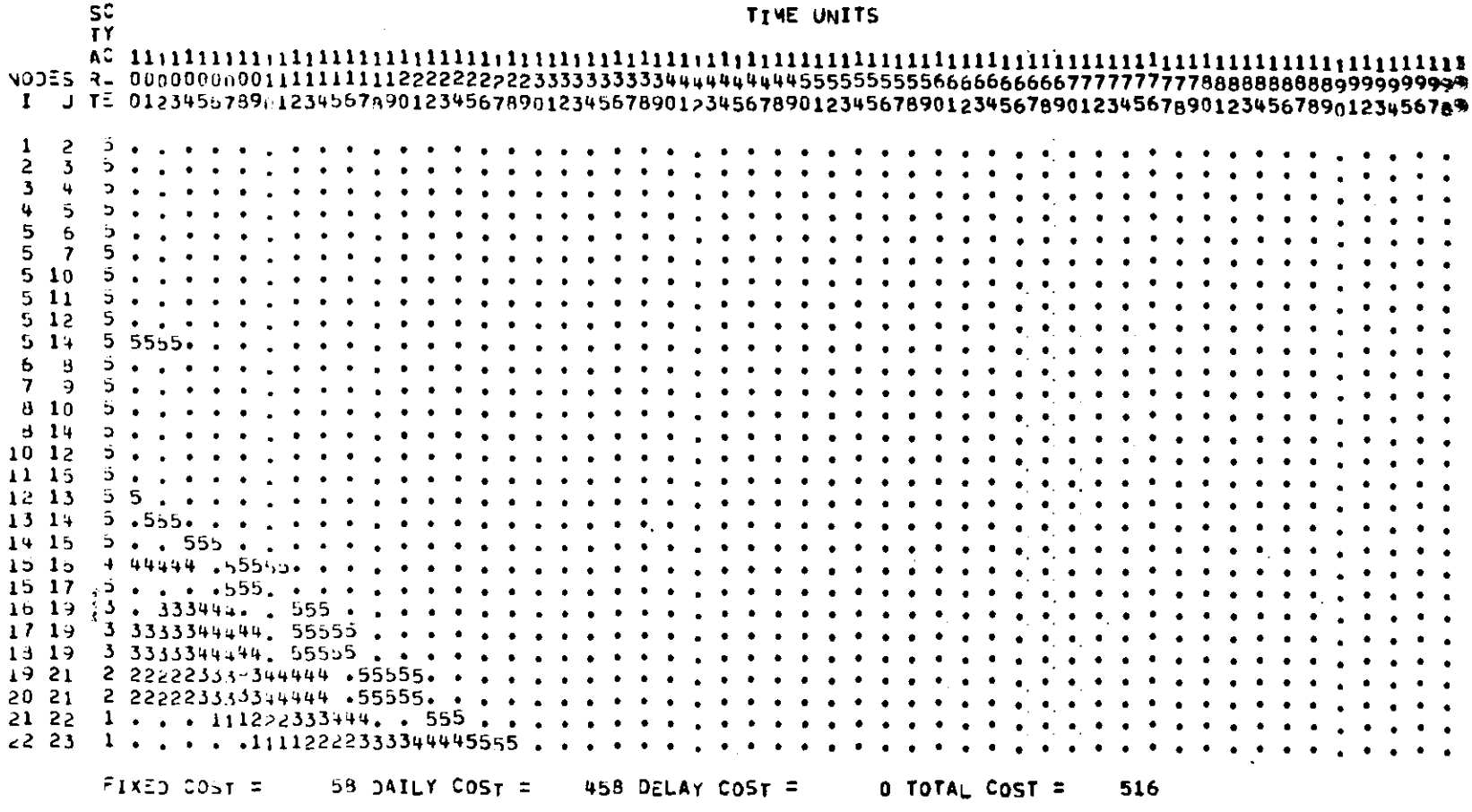


Figure 8. (Continued)

Figure 7.

These ideas on reducing activity span have resulted in the development of a very significant addition to the modified scheduling algorithm. Basically, the additional procedure consists of right-shifting activities scheduled out of the sets L and K and left-shifting the activities scheduled out of set M. A detailed step-by-step description of the procedure is given below. In order to use this procedure to reduce the span of scheduled activities, the following assumptions are made:

1. The activities are considered according to their position in an ordered set that is formed in the same manner that set R was in the scheduling procedure; i.e., by increasing order of the sum of i-node plus j-node.

2. Continuity between cycles of activities will be maintained and never destroyed. Therefore, the procedure may require the use of a "block" of cycles, which is defined as a series of cycles which are scheduled continuously. For example, activity (16,19) in Figure 7 has three blocks consisting of cycles one and two, cycles three and four, and cycle five. A continuous activity is considered to be one block.

3. For activities which were scheduled out of L or K and which are not continuous, the block containing the last cycle will never be right shifted. This is to insure that the span of the activity will be reduced.

4. For activities which were scheduled out of the set M and which are not continuous, the block containing the first cycle will never be left-shifted. This will also insure that the span of such activities will be reduced.

The statement of the procedure is:

1. Select the last activity in the ordered set which was scheduled out of sets L or K.

2. Determine the first block of cycles of the activity.

3. Determine the minimum amount of slack between the completion of each cycle in the block and the start of the corresponding cycles of all successors of the activity.

4. Right-shift the start time of each cycle of the block by the minimum slack determined in step 3.

5. If the activity is not continuous, determine the next block. If this block does not contain the last cycle, return to step 3. If the activity is continuous or the block of the activity contains the last cycle, proceed to step 6.

6. By continuing to select activities from the ordered set in reverse order, select the next activity that was scheduled out of sets L or K and return to step 2. If there are no more activities which were scheduled out of these sets, proceed to step 7.

7. Select the first activity out of the ordered set which was scheduled out of the set M.

8. Determine the last block of cycles of the activity.

9. Determine the minimum amount of slack between the start of each cycle in the block and the completion of the corresponding cycles of all predecessors of the activity.

10. Left-shift the start time of each cycle of the block by the minimum slack determined in step 9.

11. If the activity is not continuous, determine the next block. If this block does not contain the first cycle, return to step 9. If the

activity is continuous or the block of the activity contains the last cycle, proceed to step 12.

12. By continuing to select activities from the ordered set in increasing order of the sum of their i-node and j-node values, select the next activity that was scheduled out of the set M and return to step 8. If there are no more activities which were scheduled out of the set M, then the procedure is completed.

Though not an exact procedure that will result in minimal span for every activity in the network, it can be seen that this procedure will result in reduced span, wherever possible, for all activities in the network. A flowchart of the procedure is given by Figure 9.

The major features of the scheduling algorithm have now been presented and discussed. However, no use has yet been made of the assumption concerning extending the completion time of the project at a fixed cost for each day of delay. This assumption will be considered next in the extension of the algorithm to allow selecting activities in the project which must be continuous and scheduling cyclic projects at arbitrarily selected completion times.

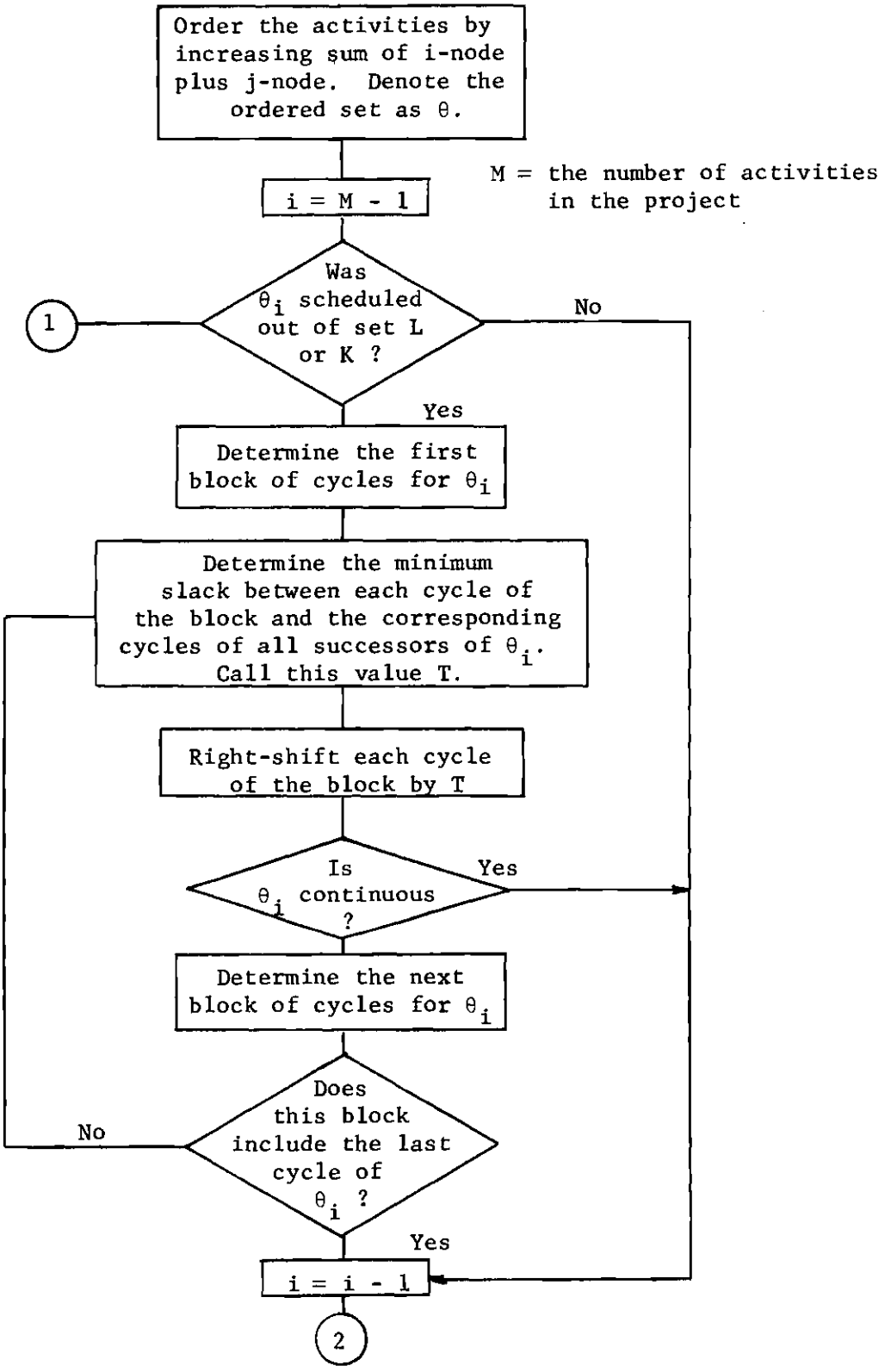


Figure 9. Procedure for Reducing the Span of Activities

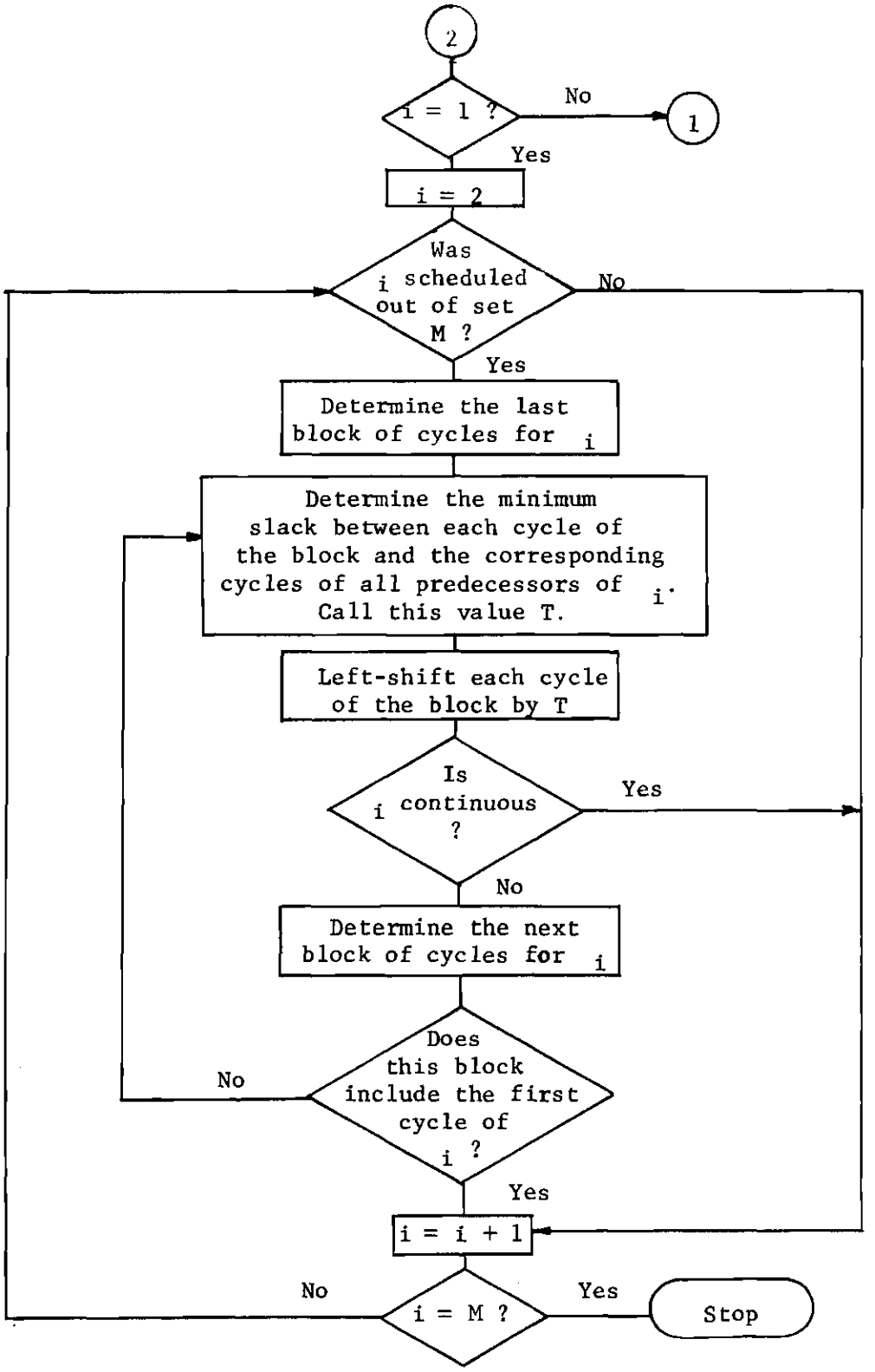


Figure 9. (Continued)

CHAPTER IV

EXTENSIONS OF THE BASIC ALGORITHM

A basic procedure for cyclic project scheduling was described in the previous chapter. The algorithm was based upon the original work of Ferraz (5) and included several modifications developed to make the procedure more efficient and effective. In this chapter, several adaptations and extensions of the algorithm are presented.

The original work of Ferraz did not allow for the introduction of the important constraint requiring any selected activity to be scheduled continuously while minimizing other interruption costs. In addition, Ferraz developed his original algorithm under the assumption that the project should always be completed at the earliest possible time. For the purposes of this research, however, this assumption was not made. Instead, it has been assumed that the project may have its completion date extended beyond the minimum completion date at some known fixed cost per day of delay. These two ideas will now be employed in this chapter for the development of an extended scheduling algorithm for cyclic projects.

Selecting Continuous Activities

It is important to realize that the scheduling procedure seeks to maximize the continuity of the activities in a cyclic project by reducing the value of some specified objective function. However, there is no guarantee that any particular activity will be continuously scheduled throughout all cycles, unless it is an initial or terminal activity or a

pacing activity. Note that in Figure 8 the schedule yielded by the procedure still has interruptions in several activities of the project. This poses the following question: if it is known that certain activities in the project must be continuous (due to technical or economic circumstances), how is a maximum continuity schedule under such constraints obtained?

Though it may appear at first glance that this is a difficult problem to solve, a rather simple procedure has been developed that will allow such scheduling. However, it must be realized that to obtain such schedules it may be necessary to extend the project completion date. For some activities in the project, it is not at all possible that the schedule of all the cycles of an activity will ever be continuous when the project is scheduled for completion at its earliest time. Therefore, the project must be extended to allow the creation of enough slack between the start times of the cycles of an activity and its predecessors and successors in order to obtain a continuous schedule. The incurrence of additional costs due to the delay of the project must then be considered, as well as the costs of interruption in the activities of the project.

The method for achieving continuity of selected activities is summarized as follows:

1. Select the next activity in the ordered set formed by the increasing sum of i-node plus j-node (set R).

2. Compute the ESS schedule for the current activity as defined earlier. If it is not desired that this activity be continuous, return to step 1; otherwise, proceed to step 3.

3. Check to see if the ESS schedule of the activity is continuous. If it is, return to step 1; if not, proceed to step 4.

4. Schedule the last cycle of the activity at its ESS time. Then, working from the last cycle to the first cycle, schedule each cycle in close continuity with respect to its succeeding cycle. Mathematically, this is:

$$S_{i,N} = ESS(i,N) ,$$

$$S_{i,j} = S_{i,j+1} - D_i \quad \text{for } j = N-1, N-2, \ldots, 2, 1.$$

5. Set the ESS and LSS schedule of each cycle to its continuous schedule:

$$ESS(i,j) = LSS(i,j) = S_{i,j} \quad \text{for } j = 1, 2, \ldots, N-1, N.$$

6. Remove the activity from the set of unscheduled activities, the set R, and return to step 1.

It can be seen that this procedure will make any activity continuous which must have no interruptions and will also insure that the project will be completed as soon as possible under such a constraint. Since the last cycle is scheduled at its early start time, forcing all activities to be continuous with the last cycle will insure that the continuity constraint is satisfied and the project is not being delayed any longer than necessary.

This modification may be incorporated into the cyclic project scheduling procedure. The six step procedure outlined above is substituted for the step of computing the ESS times of the activities in the project. If no activities are required to be continuously scheduled, the regular ESS schedules for all activities will result. If any activities in the project are required to be continuous, the procedure will schedule these selected activities without allowing interruptions in their schedules. The remaining unscheduled activities will have their ESS and LSS times determined in the normal manner. The remainder of the scheduling procedure may then

be applied to the activities in R and a maximum continuity schedule will be obtained.

As an example, the extended procedure was applied to the sample problem of Figure 3. It was arbitrarily decided that activity (4,5) should be continuous and that the delay cost would have a value of eight. The Gantt chart representation of the resulting schedule is given in Figure 10. It should be noted that, in forcing this activity to be continuous, it is necessary to extend the project completion date considerably beyond that of the schedule in Figure 8, which is the schedule for the minimum completion time. However, it is also important to note that many activities which succeed activity (4,5) are also continuous. Even though there has been an increase in costs due to the delay of the project, significant cost reduction with respect to activity (4,5) and many of its successors has also occurred. This implies that there may be considerable advantages in scheduling activities to be continuous where the costs of delay are not as significant as the costs of interrupting some activities. Thus, it may actually be possible to find a schedule with greater continuity and lower total costs than may be obtained at the minimum completion time. This idea becomes more important when studying the use of the algorithm with selected alternate completion dates.

Selecting Alternate Completion Times

Rather than selecting activities to be continuous and studying resulting schedules, it may be desirable to simply pick a completion time

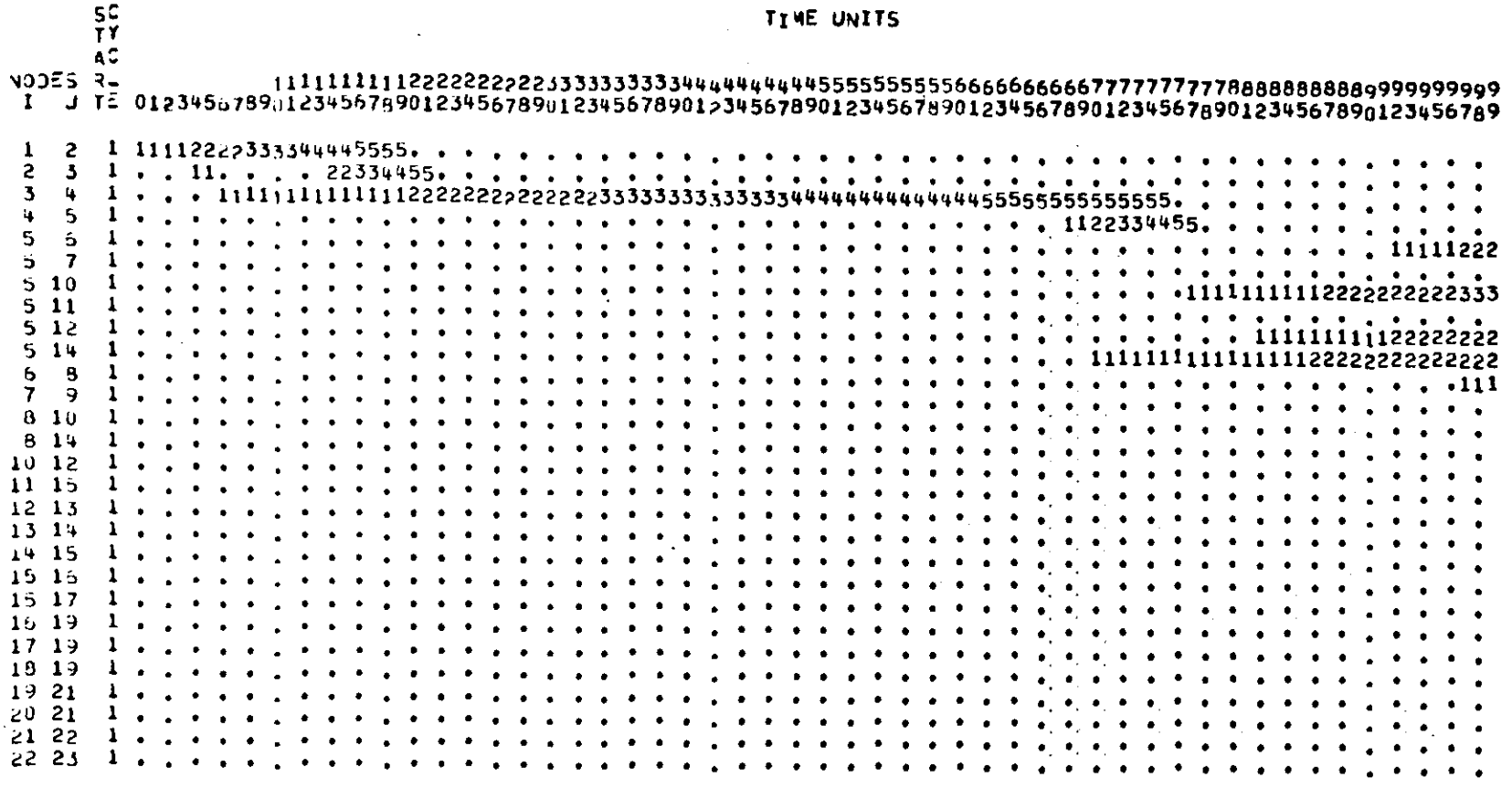


Figure 10. Schedule for the Project of Figure 3 Where Activity (4,5) Is Selected for Continuous Scheduling

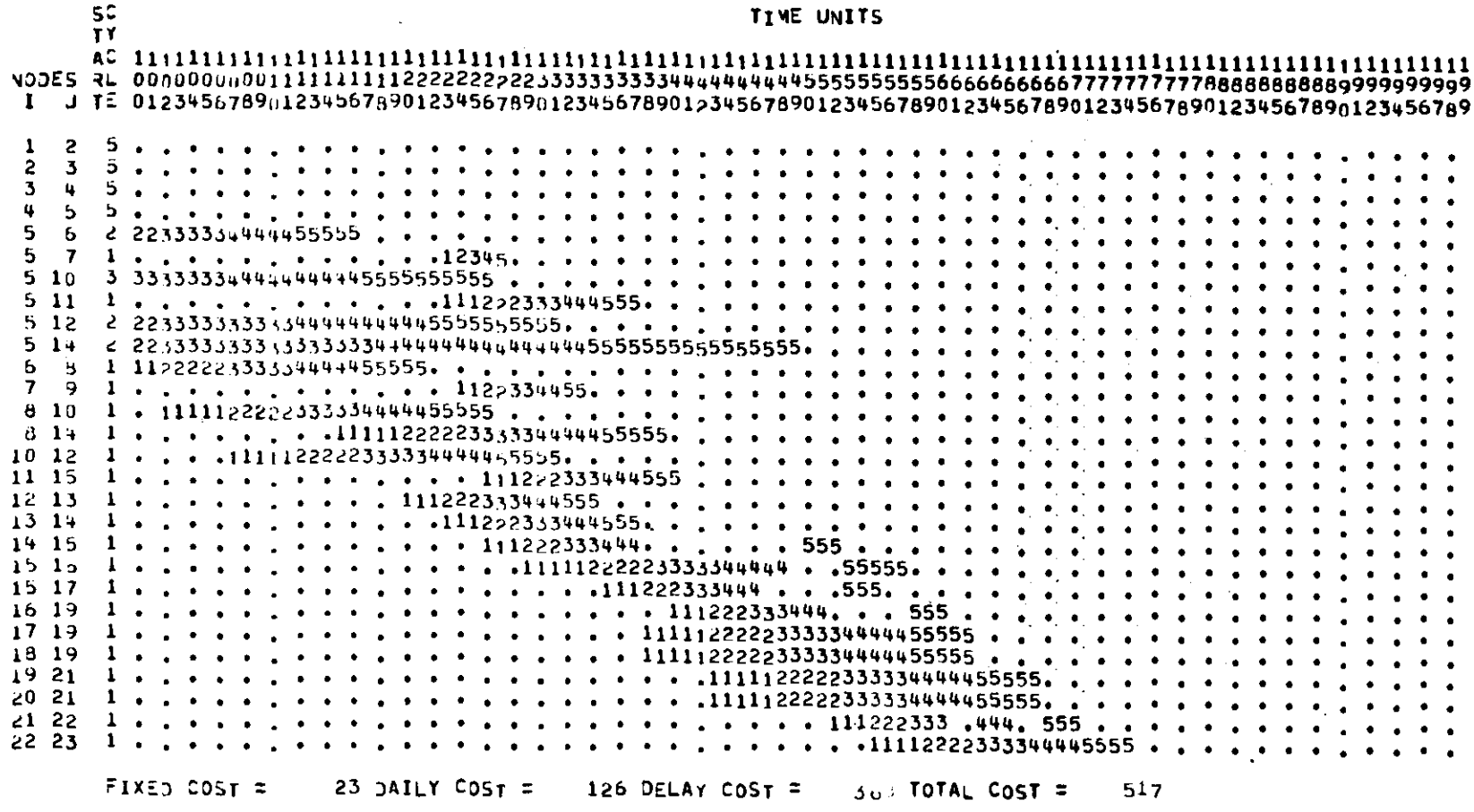


Figure 10. (Continued)

and investigate the schedule which then results from the application of the scheduling algorithm. Before doing so, however, care should be taken in the selection of the alternate completion time, for there are both upper and lower bounds on its value.

The lower bound on completion time, of course, is the minimum completion time determined from the computation of early start times. The determination of an upper bound requires the application of the procedure discussed above for selecting continuous activities. In this case, if all activities were selected to be continuous, the "maximum" completion time schedule would be obtained. Though it would be possible to obtain feasible schedules with even greater completion times, it would be of no advantage, since all activities could be continuous and there would be an increasing project delay cost without any offsetting benefit of decreased project interruption costs. Therefore, the completion time of a schedule with no activity interruptions is equivalent to the upper bound on project completion time.

As a modification of the scheduling procedure, it would first be necessary to determine the minimum completion time from the early start time computations and the maximum completion time from the schedule with all activities continuous. With the upper and lower bounds defined, a completion time may then be selected. The late start times could then be recalculated based upon this new completion time and then the modified scheduling procedure for cyclic projects could be applied, using the previously calculated early start times and late start times.

As an example of the use of this procedure, it was applied to the

problem of Figure 3. The minimum and maximum completion times determined were 129 and 201, respectively. A completion time of 149 was selected arbitrarily and was used to produce the schedule of Figure 11. Using the objective function of Equation (9) to compute total costs, the value for the schedule of Figure 11 is found to be 395, significantly less than the total cost of 516 obtained at the minimum completion time of 129, as shown by Figure 8. Thus, it appears that there may be situations where it would be possible to extend the project completion time and reduce total project costs.

As a further investigation of the potential for reducing project costs by extending the completion time, the cost was computed for schedules of the project of Figure 3 at five unit intervals from 129 to 201: at 129, 134, 139, . . ., 194, 199, and 201. The resulting costs are represented in the graph of Figure 12. The graph shows that there does appear to be a "minimum" cost and that it occurs somewhere around the completion time of 149.

For this very simplified example, it might appear that there is a unique minimum cost for the cyclic project schedule and that this may be easily found by a search-oriented procedure adapted from the operations described to get the project cost curve of Figure 12. Though the example does demonstrate the idea that it may be possible to get a lower cost by allowing a delay in the project, it may not be assumed that there is only a single minimum cost or that a total cost curve like that of Figure 12 will always result. The shape of the total cost curve and the appearance of an absolute minimum will depend upon the objective function, the assumptions made, and the cost parameters of the project itself.

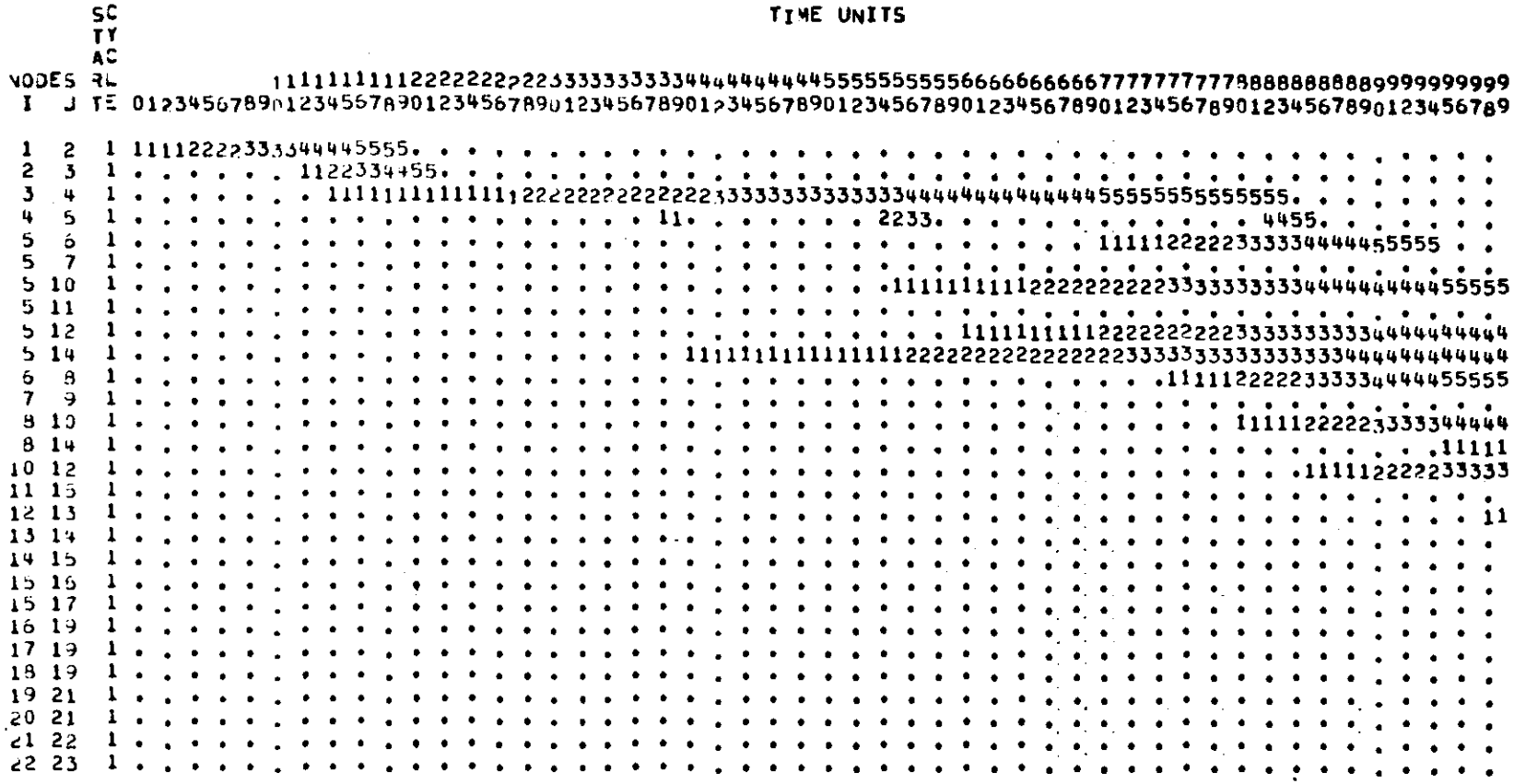


Figure 11. Schedule for the Project of Figure 3 With a Selected Completion Time of 149

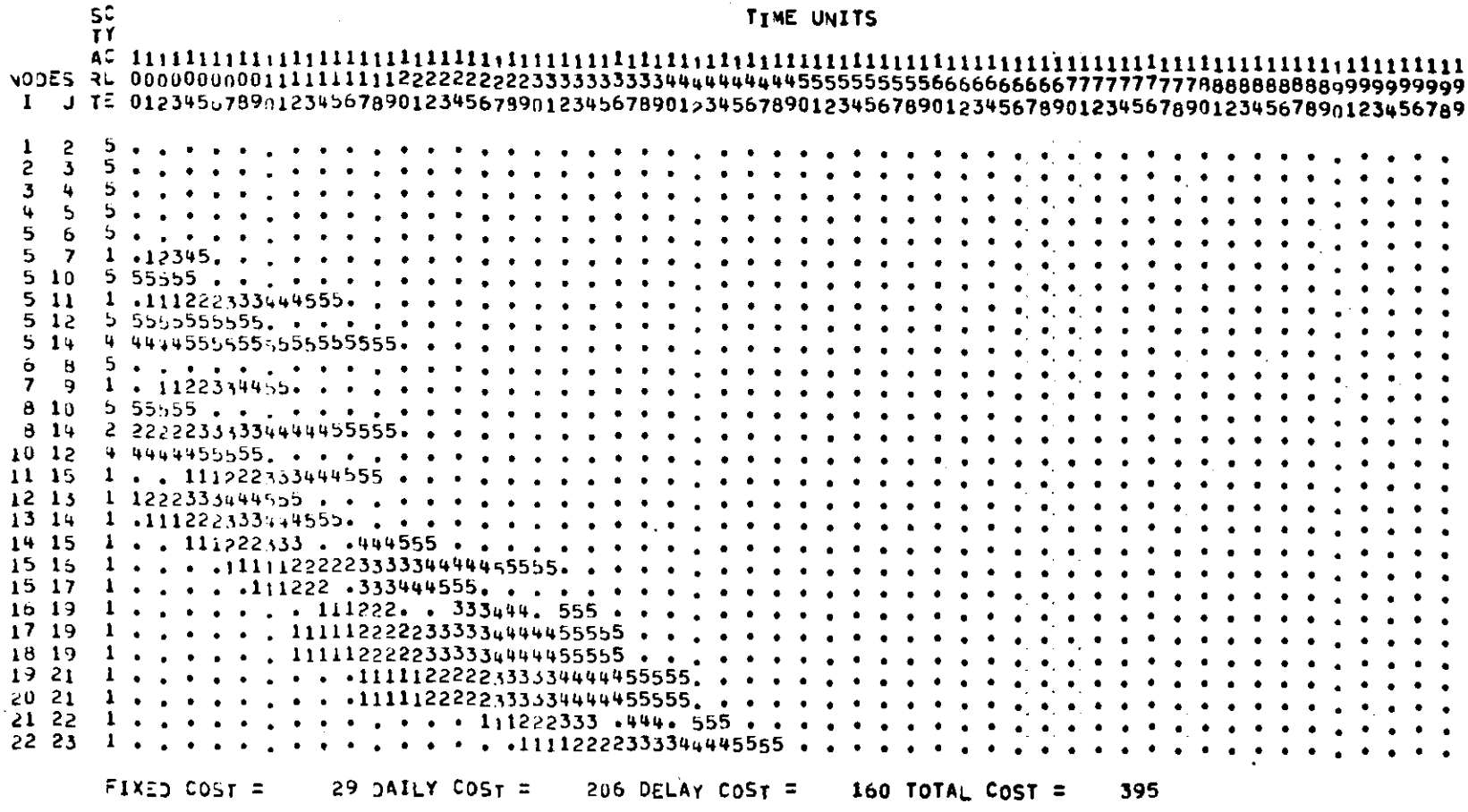


Figure 11. (Continued)

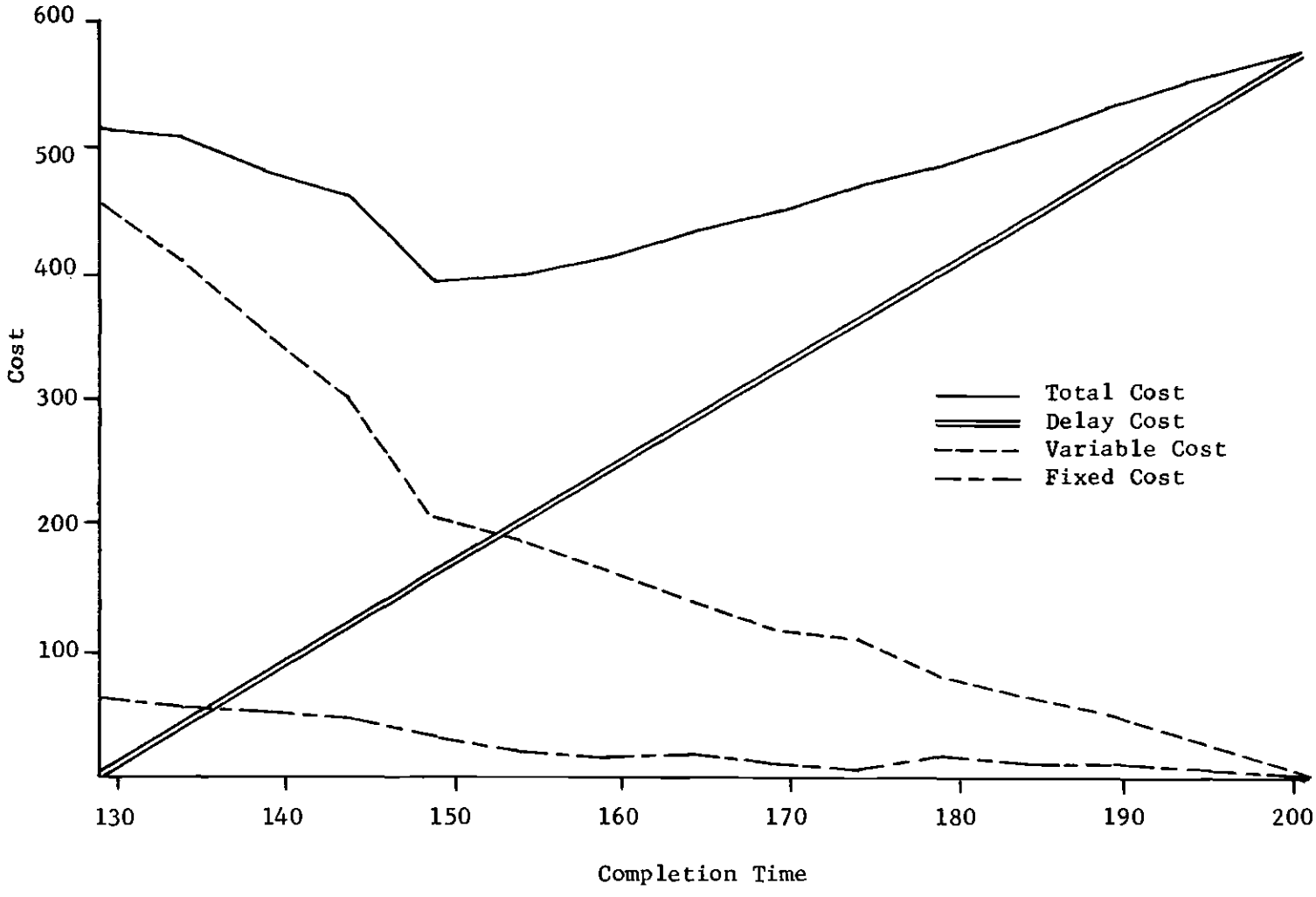


Figure 12. Graph of Total Project Costs for Feasible Completion Times for the Project of Figure 3

As an example of the effect of the cost parameters on the total cost of the project, consider a curve like that of Figure 13. Here, there is one activity in the project which has one interruption with a fixed cost of interruption that is very large in relation to the other fixed costs of the project. When this interruption can first be removed at some completion time of the project which is greater than the minimum completion time, a very significant step decrease in the fixed cost curve occurs. When summed with the other two cost curves, a total cost curve results which has two relative minima. The minimal cost would then be the lower of these two total costs. This situation did not occur in the example of Figure 12 because all of the fixed interruption costs were relatively small and relatively close to each other. When applying the procedure at different completion times, it is also not safe to assume that the total fixed costs of interruption will decrease. As an example, consider the change when going from a completion time of 174 to a time of 179 in Figure 12. The fixed interruption cost shows a slight increase. Thus, care must be taken in computing cost curves for projects that have wide variability in fixed interruption costs among different activities and where the fixed costs are being evaluated by the objective function. It may be necessary to investigate the resulting total cost for every completion time between the minimum and maximum in order to determine the lowest obtainable cost using the scheduling procedure. This could be a costly and time-consuming procedure in itself.

An analysis of the importance of the variable interruption cost curve in the computation of total costs, however, shows some promise of

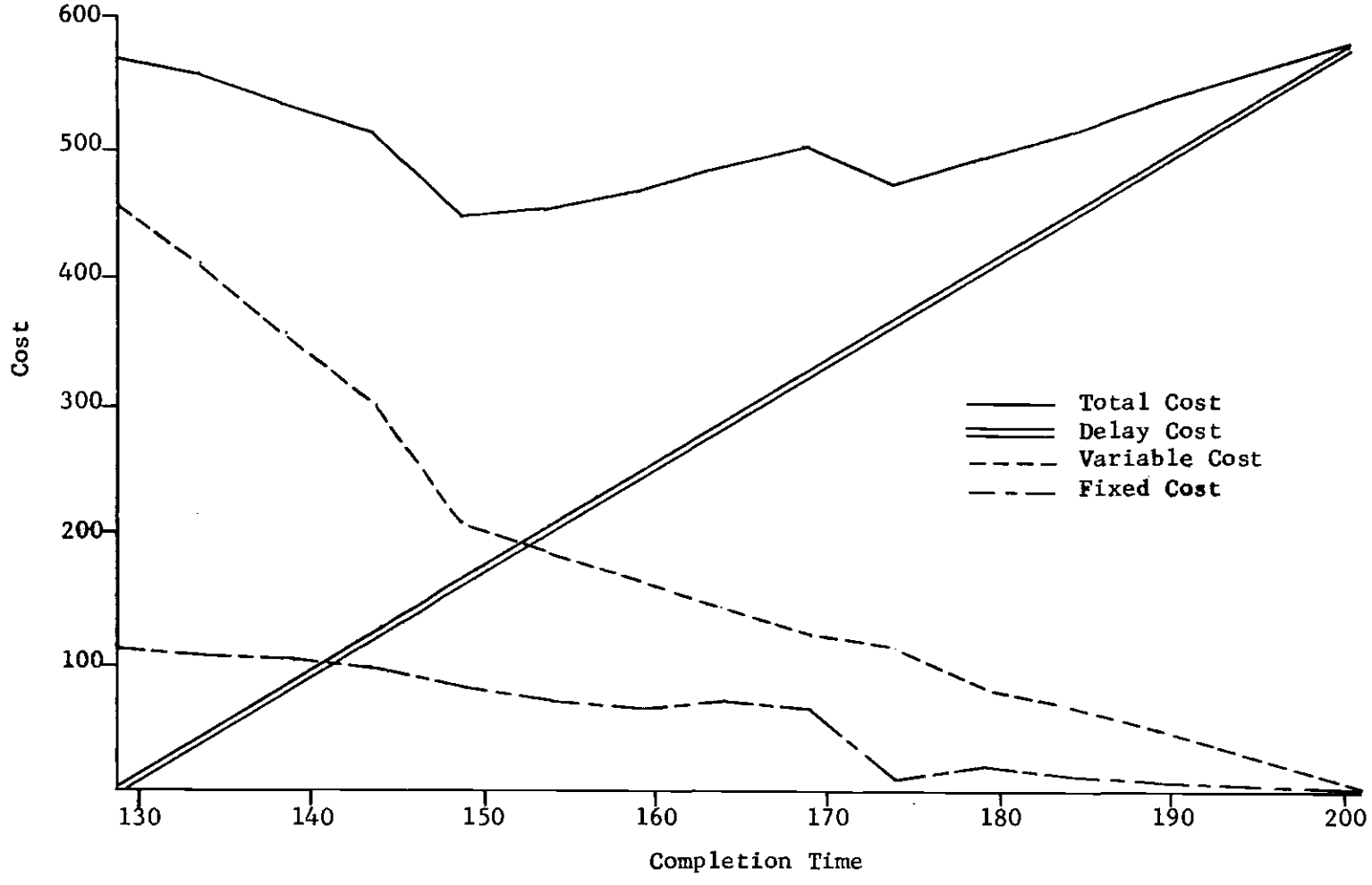


Figure 13. Resulting Project Cost Curve When There Is a Significant Fixed Interruption Cost

leading to the development of a procedure that is much more efficient in finding a low cost schedule for a cyclic project. Suppose that fixed costs of interruption were not being considered; that is, the objective function being reduced is that consisting solely of the sum of variable interruption costs and project delay costs. For the project of Figure 3, the resulting cost curve over the completion times between 129 and 201 is given by Figure 14.

The important information to be obtained from this graph comes from an analysis of the variable cost of interruption curve. It can be seen that, while going from the minimum completion time to the maximum completion time, this curve represents a cost that is always decreasing. This is not surprising since for every unit increase in the completion time of the project it is always possible to decrease the span of at least one activity in the project, thus decreasing the variable interruption cost. As the completion time is extended, the scheduling procedure has available increased slack for all cycles of all activities, and this increased slack is used to decrease the length of interruptions in activities in the project (and thus the variable interruption cost), as is clearly demonstrated by the cost curve.

Further analysis shows, however, that the cost curve obtained by simply utilizing different completion times in the scheduling procedure which uses an objective function based on the sum of variable interruption and project delay costs may not give the lowest obtainable total cost. Consider the fact that the variable interruption cost curve of Figure 14 is not a piece-wise linear decreasing (or convex) curve; i.e., the absolute value of the slope of the curve at a time t_1 is not necessarily greater

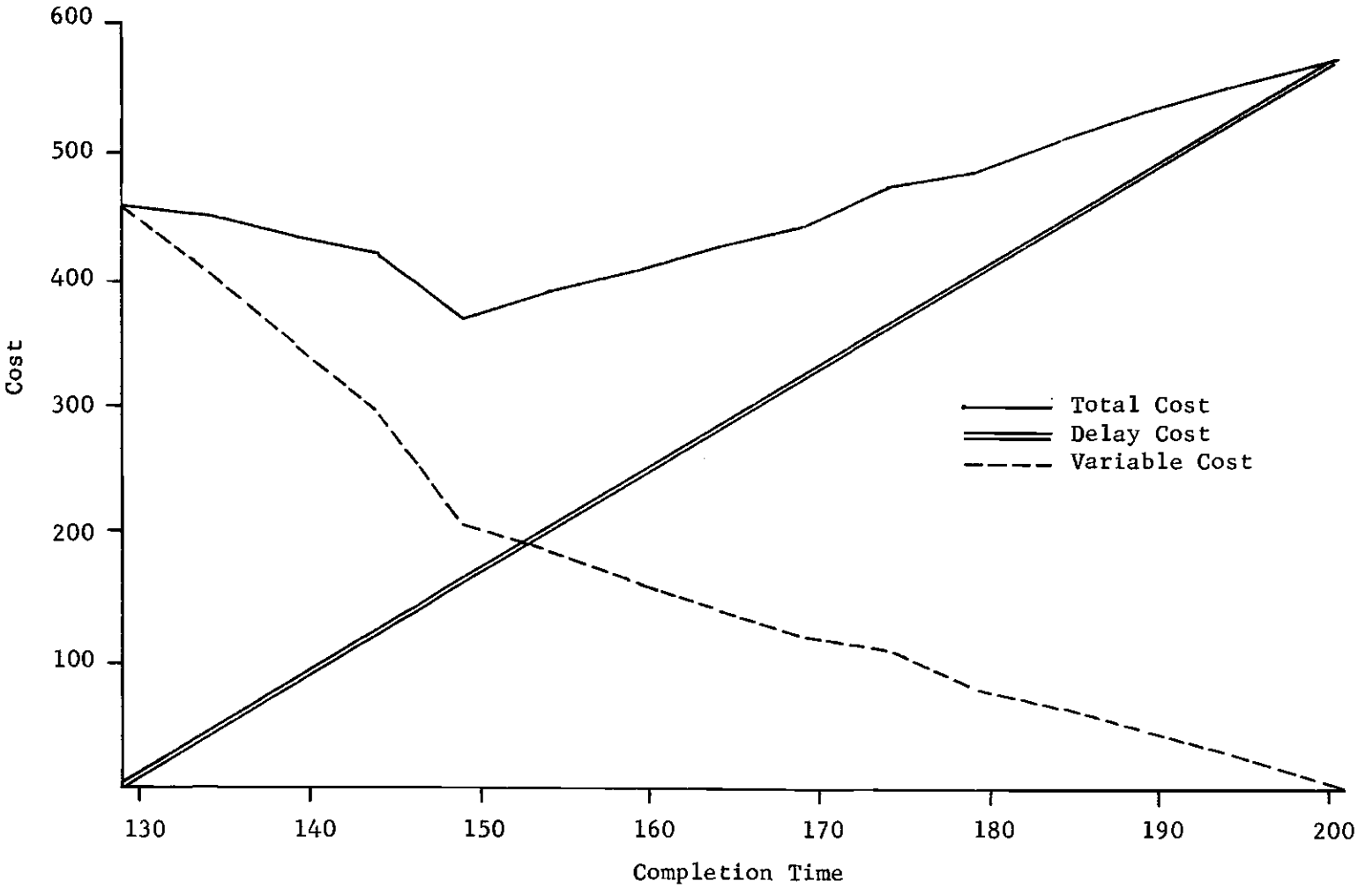


Figure 14. Graph of Delay and Interruption Costs for Feasible Completion Times for the Project of Figure 3

than or equal to the absolute value of the slope of the curve at a time t_2, when t_2 is greater than t_1. The effect of this is highly significant. If the curve were a convex curve, it is quite possible that the sum of the interruption cost curve and the linear delay cost could yield a lower total cost than that obtained from the scheduling procedure. An example of such a curve is given in Figure 15. This shows that if it were possible to extend the project completion date, such that the interruptions with the most significant daily interruption cost were always being reduced in span, then for any completion time a lower interruption cost could be obtained. When combined with the daily delay cost, this could give a lower total cost, at an earlier point in time, than that obtained by a search-oriented procedure. Therefore, a steepest-descent approach with respect to reducing variable interruption costs would possibly yield a better solution.

The conditions for which the application of the scheduling procedure for extended project completion dates may not provide the best cost curve for the variable interruption costs can be determined by considering the steps of the procedure as outlined in Chapter III. Since the procedure works toward the "middle" of the network, it is apparent that the activities which are receiving the benefit of the increased slack are the activities at the "front" and "back" of the network. As the procedure works toward the unscheduled activities that remain, there is less and less slack available for reducing the length of interruptions. This characteristic of the scheduling procedure can be observed by considering the Gantt chart of the schedule of Figure 11, which has a completion time of 149, and the minimum completion time schedule of Figure 8.

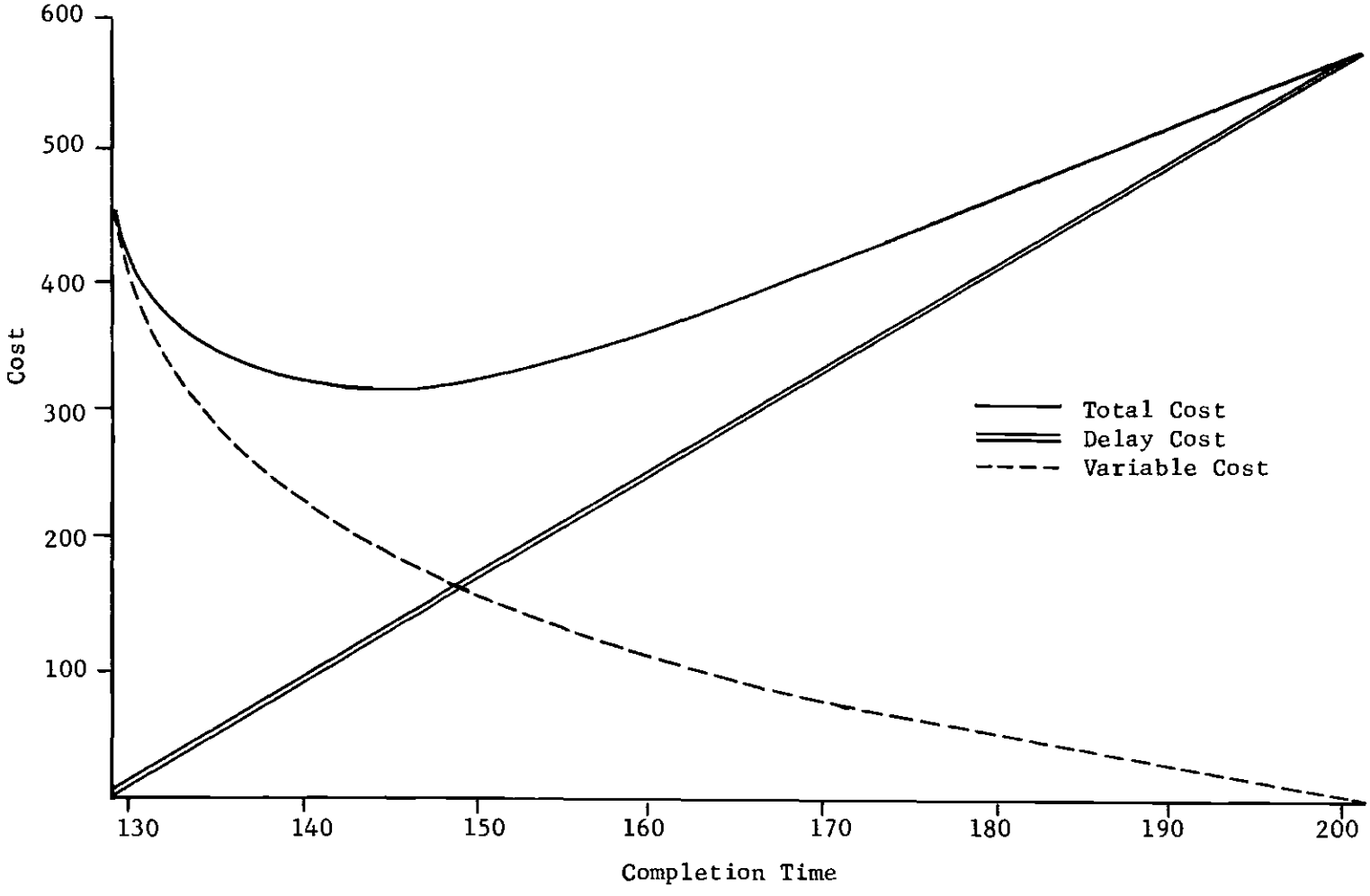


Figure 15. Example of a Convex Interruption Cost Curve

The effect of this characteristic of the scheduling procedure is that the activities with the highest interruption cost may not have their spans reduced. There may be activities toward the middle of the network that have high interruption costs and which could significantly reduce the total variable interruption cost for a given increase in the project duration. However, their unscheduled predecessors and successors must be scheduled before they become candidates for scheduling. These predecessors and successors may not have as high interruption costs, but they will be the first to make use of the available slack that has been created. Thus, this aspect of the scheduling procedure will result in the cost curve of Figure 14.

This analysis implies that there is a need for a better approach for finding a low-cost schedule when it is possible to delay the completion of the project. Particularly, there is a need for a steepest-descent time-cost trade-off procedure. The development of this procedure is considered in the next chapter.

CHAPTER V

A STEEPEST-DESCENT TIME-COST TRADE-OFF PROCEDURE

In the previous chapter, it was demonstrated that it is possible to use the modified scheduling algorithm to schedule cyclic projects where the project completion time was extended beyond the minimum. It was also demonstrated that it may be possible to reduce total costs of a project by extending the project completion time to such a point that the reduction in interruption costs for cycles of activities in the project outweigh the resulting increase in the project delay cost. This suggests that there is a trade-off occurring, where an extension of completion time may be "bought" at a rate cheaper than the allowance of costly interruptions in schedules of activities.

The preceding chapter discussed briefly the problem that the application of the procedure at different completion times may not actually yield the lowest possible costs. Particularly with respect to the assumption that fixed interruption costs are not being considered, it was suggested that there might be a better method to obtain the lowest possible sum of delay costs and variable interruption costs. This method will now be developed and presented.

An Approach to the Time-Cost Trade-Off Problem

As stated earlier, the problem considered here is one in which it is possible that additional time to complete the project may be obtained at some fixed cost per unit of time that is low enough so that costly

interruptions may be reduced in length, resulting in a lower total cost. It was pointed out in the previous chapter that there were aspects of the modified scheduling algorithm which might not necessarily allow its application to the scheduling of projects at various completion times in order to find the lowest attainable cost under the assumption of a time-cost trade-off. It might also be pointed out that in very large projects, with a significant number of cycles and a wide range over which to select alternate completion times, it could be quite time consuming and costly to reapply the procedure for every known possible completion time in order to search for a low total cost. Thus, there is motivation not only for a procedure that will produce the lowest attainable total cost, but for one that will be highly efficient in doing so.

Basically, the algorithm which will be developed here is one that makes use of a steepest-descent approach for reducing activity interruption costs. Since variable costs of interruption are expressed in cost per unit of time, and fixed costs of interruption are not considered, it will be possible to make a direct comparison between costs of reducing activity interruptions and increasing the completion time of the project. If, for increases in the project completion time, it is possible to select the activities from the current schedule of the project which may have their spans reduced and result in the most significant cost decrease, it will be possible to obtain a curve like that of Figure 15. The major development in this approach, therefore, will be to determine what those activities are and the conditions under which they may have their spans reduced in order to reduce total cost.

The developed procedure will begin by using the schedule obtained

for the project at its minimum completion time using the algorithm of Chapter III, with the objective function composed of the sum of total variable interruption cost as given by Equation (6). This is the only time at which the scheduling algorithm will be used in the time-cost trade-off procedure. The remainder of the procedure for time-cost trade-off will be a multi-step procedure which considers, at every iteration, subsets of the activities in the project which may have their spans reduced for an increase in the project duration. Based upon a comparison of the variable interruption costs of the activities in each subset and the delay cost for the project, it will be possible to select a subset which provides for a maximum decrease in project interruption costs for each day's increase in delay costs. A right-shifting procedure for reducing activity span, similar to that developed in Chapter III, may be used to reschedule the activities so as to reduce their span. When the activities in the appropriate subset have been rescheduled, the subsets are reformed and the procedure starts over.

The development of the time-cost trade-off procedure will now be presented. Each of the basic steps will first be discussed in detail and will be followed by a discussion of the application of the procedure.

Bounding Activities and Bounded Sets

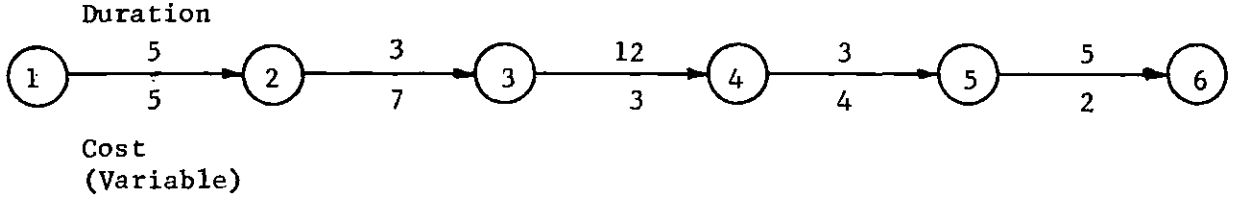
The first step of the time-cost trade-off procedure requires the identification of distinct subsets of activities in the project. The formation of these sets is the single most important development in this scheduling procedure. In order to motivate a discussion of how these sets are determined and used, an example will be given.

Suppose it is necessary to find a low-cost schedule for five cycles of the project of part (a) of Figure 16. The algorithm of Chapter III is applied and the schedule of part (b) of Figure 16 results. This is the minimum completion time schedule for the project.

Next, the extension of the project completion time is considered in order to reduce the interruptions that remain in activities (2,3) and (4,5). The procedure that is to be used for reducing the span of the activities is analogous to that of the span-reduction for the activities scheduled out of the sets L and K as presented earlier. As the schedules of successor activities are delayed, the minimum slack between the blocks of cycles of an activity and the successor is determined in order to reduce the span of the activity. As an example, suppose that the completion time of the project is increased one time unit by delaying the start time of each cycle of activity (5,6). This creates one unit of slack between cycle one of activity (4,5) and cycle one of activity (5,6). Thus, as shown in part (a) of Figure 17, it is possible to delay the start of cycles one, two, three, and four of activity (4,5) by one time unit and thereby reduce the span of the activity by one.

The next step is now to try to reduce the span of activity (2,3), the only other interrupted activity, by one time unit. However, an examination of the schedule in part (a) of Figure 17 shows that this will not be possible. As assumed under the right-shifting procedure for reducing the span of activity (4,5), the last cycle of the activity was not rescheduled so that the span of the activity could be reduced. The predecessor activity, activity (3,4), is continuous and there is no slack between the last cycle of activity (3,4) and the last cycle of activity (4,5).

(a) Project Network for One Cycle



(b) Five-Cycle Project Schedule

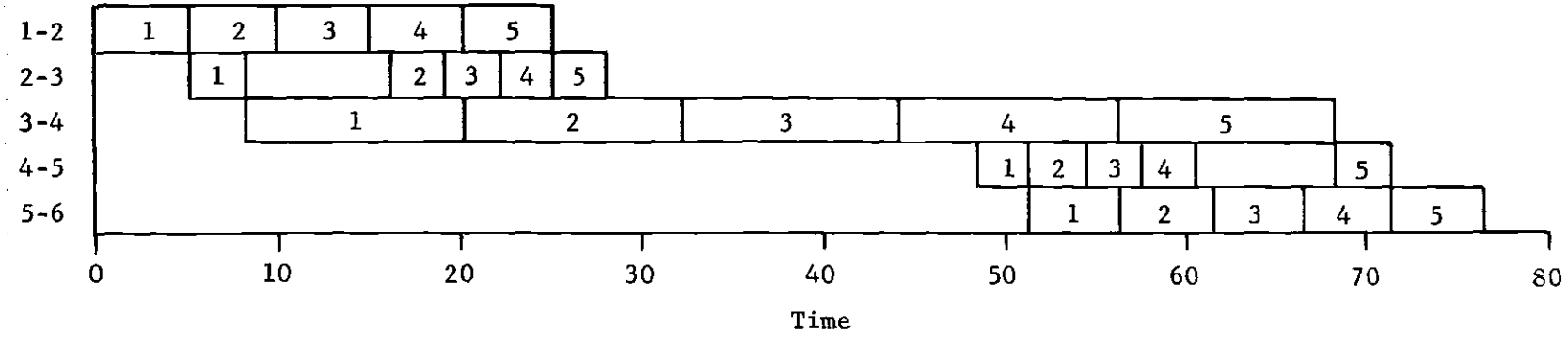


Figure 16. Network and Schedule for a Five-Cycle Project

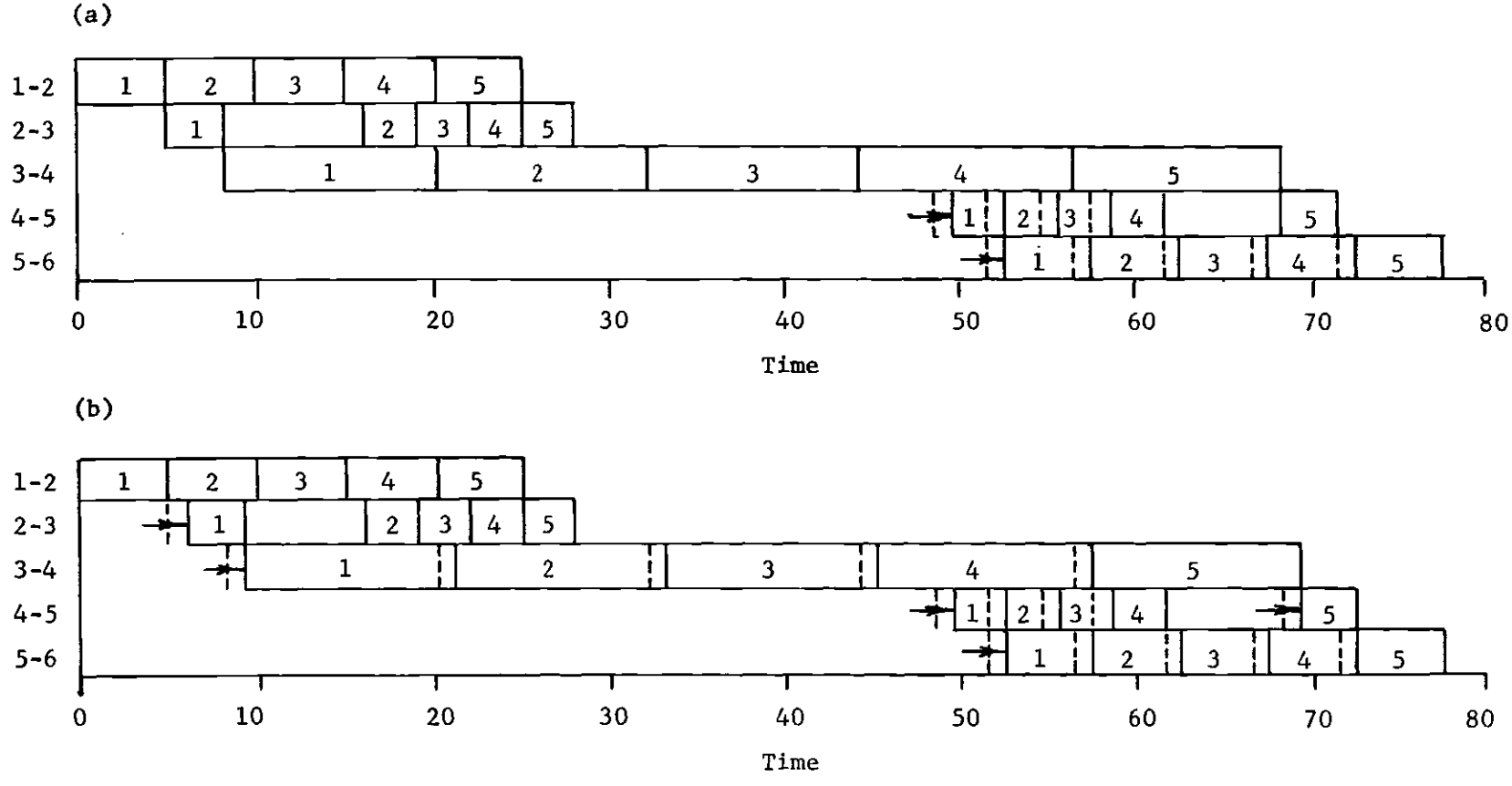


Figure 17. Reducing the Span of Activities in the Schedule of Figure 16

Therefore, it is not possible that the cycles of the continuous activity (3,4) may be delayed by one time unit. Since the first cycle of (2,3) has zero slack with respect to the first cycle of (3,4), it is not possible to reduce the span of activity (2,3). Therefore, if the span of activity (4,5) is reduced when the completion time is increased by one time unit, the slack made available to (4,5) will not be available to the other interrupted activity in the schedule.

The problem encountered above is quite significant. In Figure 16, part (a), it may be noted that activity (2,3) has an interruption cost of 7 while activity (4,5) has an interruption cost of 4. Thus, by reducing the span of (4,5) when the completion time is extended, the lowest cost for the resulting completion time is not being obtained. A lower cost could be obtained by reducing the span of activity (2,3). Therefore, as shown in part (b) of Figure 17, the proper rescheduling of activities with the increase of one time unit in the schedule of the project is given by span reducing activity (2,3) by one time unit and delaying the start of all cycles of all successors by one time unit.

The example provided by Figures 16 and 17 has demonstrated some very important concepts for rescheduling cyclic projects with extended completion times. The first important concept is that, when increasing the project completion time, it will not be possible to reduce the span of every interrupted activity in the schedule of the project. Instead, there will be some well-defined subsets of interrupted activities which are associated with some very distinct continuous activities. It will only be possible, when increasing the project completion time, to reduce the span of activities in certain subsets. These concepts, as demonstrated

by the example, have resulted in the definition of two important terms for the time-cost trade-off process: bounding activities and bounded sets.

A bounding activity is one like activity (3,4) in Figures 16 and 17. It can be seen that this activity is continuous and that its cycles may be delayed in starting by one time unit only if the last cycle of all of its successors (activities (4,5) and (5,6)) have their last cycles delayed by one time unit. In the case where activity (4,5) was span-reduced when the project completion time was extended, the last cycle of activity (4,5) was not delayed; therefore, no cycle of activity (3,4) could be delayed, also resulting in no chance for span reduction of activity (2,3). Thus, activity (3,4) placed a "bound" on reducing the span of activities in the project. In light of this, the following definition has resulted: a bounding activity is any continuous activity in the schedule of the project which cannot have all of its cycles delayed without delaying the last cycle of all activities on a single path which succeeds the activity.

The reasoning behind the definition and the occurrence of bounding activities can be better demonstrated by considering in somewhat more depth the schedule of part (b) of Figure 16. Ferraz (5) has demonstrated that a cyclic project schedule, when scheduled at the minimum completion time, has the following property: the critical path of the project is formed by the first cycle of all activities on the longest path of a single cycle which precede the pacing activity; all cycles of the pacing activity; and the last cycle of all activities which lie on the longest path of a single cycle and which succeed the pacing activity. By applying this property to the schedule of Figure 16, since there is only one path, it is noted

that the critical path is formed by the first cycle of activities (1,2) and (2,3), all cycles of activity (3,4), and the last cycle of activities (4,5) and (5,6). Because of this property, it should be quite obvious why activity (3,4) would be a bounding activity: the schedule of its last cycle is directly dependent of the last cycle of all activities on the critical path. Therefore, only if all of the last cycles on the critical path are delayed can activity (3,4) be delayed. Also, since activity (1,2) has its first cycle on the critical path, this dependency is transferred to activity (1,2) and it will be a bounding activity as well. Activity (5,6) could be considered a bounding activity also, because delaying its schedule is dependent on delaying the completion of the project which, in a sense, is analogous to delaying the last cycle of a successor activity.

As demonstrated in the example of Figures 16 and 17, there will also be very distinct subsets of activities which may have their spans reduced when the project completion time is delayed. These subsets result from the occurrence of the bounding activities and will be defined as follows: a bounded set is a subset of the activities in the project which is associated with a bounding activity and which is composed of all non-bounding activities which are part of a chain which precedes the associated bounding activity.

By application of the above definition, it is seen that there are two bounded sets in the example of Figure 16. For example, activity (2,3) forms the bounded set associated with the bounding activity (3,4), and activity (4,5) is the bounded set associated with the bounding activity (5,6). The use of the term "bounded" is also illustrated well here because the activities belonging to any set appear to be "bounded" by certain

continuous activities. It should also be noted that it would be possible to have more activities in a bounded set if there were a chain of non-bounding activities preceding either activity (3,4) or (5,6). As a further example of the concepts of bounding activities and bounded sets, consider the project given by Figure 3 and its resulting schedule in Figure 8.

The schedule of Figure 8 shows that there are four bounding activities in the project when scheduled at its minimum completion time. These activities are activity (1,2), activity (3,4), activity (5,14), and activity (22,23). Since activity (1,2) has no predecessors, it has no bounded set associated with it. Activity (3,4) has only one activity in its bounded set, activity (2,3), as does activity (5,14) which has activity (4,5) in its bounded set. All other activities form the bounded set for activity (22,23). It is important to note that this schedule demonstrates that any continuous activity is not necessarily a bounding activity. It is necessary that the criterion of delaying the last cycle of all successors on a path be applied.

When the bounding activities and the bounded sets of a project schedule have been identified, it occurs that, as demonstrated in the example, only certain bounded sets may have their activities span-reduced for any increase in project completion time. This can be explained as follows: if a bounding activity has all of its cycles delayed by one time unit, it will be possible to span-reduce all of the activities in its bounded set by one time unit. This is accomplished by applying the span-reducing procedure which was described earlier. Because the last cycles of the activities which are span-reduced are not shifted, then the bounding activities preceding the bounded set may not be delayed and the schedules

of their bounded sets remain unchanged. Likewise, it is not possible to span-reduce the bounded sets of all activities which succeed the delayed bounding activity, because for this delay to occur it was necessary that the last cycle of all successor activities be delayed. Also, in order to maintain feasibility of the schedules of all activities succeeding the delayed bounding activity, the schedule of each cycle of the activities must be delayed by one time unit. These properties are illustrated by the example of Figures 16 and 17.

The idea of only certain bounded sets being eligible for span-reduction is also well-demonstrated in the more complex schedule of Figure 8. It is evident that if any one of the three bounding activities with a bounded set is delayed by one time unit, it is only possible to reduce the span of the activities in the associated bounded set.

Figure 18 illustrates additional examples of the occurrence of bounding activities in project networks. For example, part (a) of Figure 18 shows two bounding activities which terminate on the same node of the network. For this configuration, it is apparent that if it were possible to delay all cycles of one bounding activity and to reduce the span of each of the activities in its bounded set, it would also be possible to shift the other bounding activity and reduce the span of the activities in its set. Therefore, this would represent the case where it would be possible to reduce the span of more than one bounded set for a given increase in the project duration. Another case is presented by the network of part (b) of Figure 18. In this case there is a chain of bounding activities, where each activity also has a bounded set. Here, it would also be possible that each of the three bounding activities could be delayed for

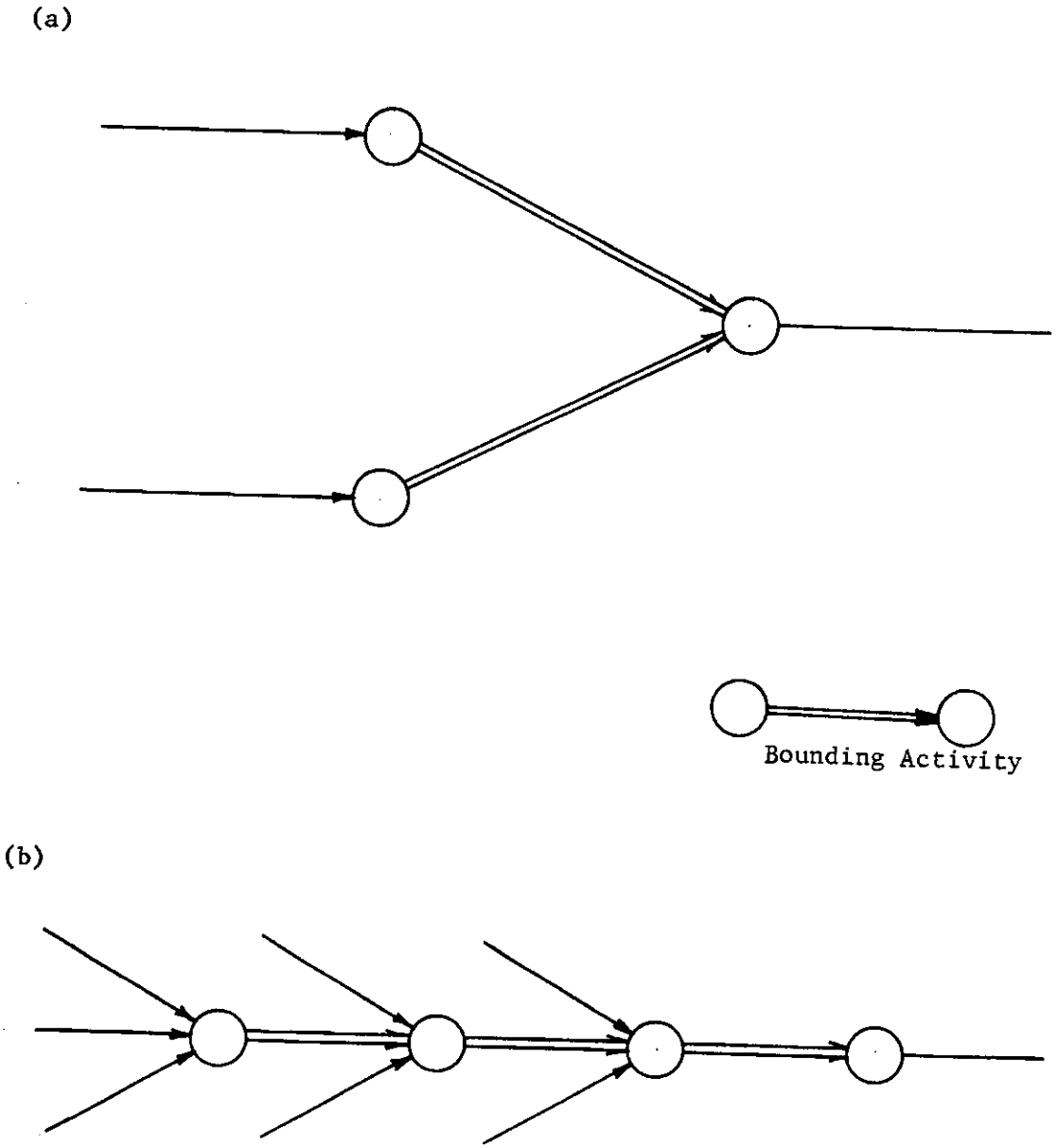


Figure 18. Examples of Common Bounding Activities

a given increase in project duration, thus resulting in the possibility of reducing the span of the activities in the bounded set of each bounding activity.

The examples in Figure 18 result in what will be defined as common bounding activities. For common bounding activities, it is possible that more than one bounding activity may be delayed. For these cases, it will be necessary to consider as the bounded set not just the set of any one of the bounding activities, but the union of the bounded sets of common bounding activities. Thus, not only will a bounded set be a subset of the activities in the project, but may, in the case of common bounding activities, also have subsets which are the bounded sets of the common bounding activities.

Now that the ideas of bounding activities and bounded sets have been defined and demonstrated in a simple example, the time-cost trade-off procedure can be developed.

Criterion for Selecting a Bounded Set

The next important concept in the development of the time-cost trade-off procedure is the development of a criterion for selecting the one bounded set of activities that is to be span-reduced when the project completion time is extended. This criterion depends only on the variable costs of interruption of the activities.

Suppose each bounded set is given a value based upon the sum of the variable interruption costs of the non-continuous activities in the set. This value would then represent the cost reduction achieved by reducing the span of the interrupted activities in the set for each unit

increase in the completion time of the project. If all of the bounded sets in the project were ranked according to their cost reduction "contribution" and the set with the largest contribution were selected, then the greatest cost reduction possible for a given increase in project duration would occur. This is the criterion that will be used. If this criterion is employed at every iteration of the time-cost trade-off procedure, this would result in a steepest-descent approach to reducing variable interruption costs; i.e., for every increase in project completion date, the largest possible decrease in variable interruption costs would occur.

Increasing Project Completion Time

After determining the bounding activities, the bounded sets, and the bounded set to which span-reduction will be applied, it will be necessary to determine how long the completion of the project should be delayed. This will be determined by examining the lengths of the interruptions in the selected bounded set.

Consider all of the non-continuous activities in the selected set. As the completion time of the project is extended unit by unit, the cycles of the bounding activity (or activities) are also shifted, allowing a unit by unit span reduction for the interrupted activities in the set. This could continue until one or more activities, which have the minimum total interruption time among all activities in the set, become continuous. The immediate effect of this would be either: (1) the activities which become continuous no longer make any contribution to reducing the interruption costs of the project, thus reducing the total contribution of the bounded set to reducing costs; or (2) the continuous activities become bounding

activities, subdividing the bounded set. Since either of these events will result in sets with decreased contributions to reducing total costs, it will only be necessary to increase project completion time by an amount equal to the minimum total interruption time of an activity in the selected bounded set.

The Time-Cost Trade-Off Procedure

The ideas developed in the previous sections on bounding activities, bounded sets, selection criterion for bounded sets, and the determination of the length of project delay can be formalized into a time-cost trade-off procedure. These ideas are now combined to give a statement of the procedure:

1. Determine the minimum completion time schedule of the project using the objective function of Equation (6).

2. Determine the bounding activities in the schedule of the project.

3. With the determination of the bounding activities, determine all bounded sets for the project schedule.

4. Compute the contribution of each bounded set by summing the variable interruption cost for each non-continuous activity in the set.

5. Select the bounded set with the largest contribution value. If this value does not exceed the per unit of time project delay cost, stop.

6. For the selected bounded set, determine the length of project delay, T, by the minimum total interruption time of an activity in the set.

7. Delay the start of the bounding activity (or activities, in the case of common bounding activities) and all its successor activities by T units, the length of project delay determined in step 6.

8. Apply the span reducing procedure to all activities in the bounded set. Return to step 2.

A flowchart of this procedure is shown in Figure 19.

Application of the Time-Cost Trade-Off Procedure

This procedure when applied to the minimum completion time schedule of any cyclic project will result in a lower total cost for the project. This is evident from considering the application of the procedure to a cyclic project schedule.

When the minimum completion time schedule has been obtained using the objective function of minimizing variable interruption costs (step 1), the bounding activities and bounded sets may be determined as described in steps 2 and 3. The contribution values of the bounded sets are then computed (step 4) and the set with the greatest potential for reducing costs is selected and the procedure stops if no value is greater than the delay cost (step 5). With the determination of the length of project delay (step 6), the project completion time is then extended and the appropriate activities in the selected bounding set are reduced in span (steps 7 and 8). The procedure is then used to reform the bounded sets and the steps of the algorithm are replicated once again.

As the spans of activities are reduced in the selected bounded set at each iteration, one of two events may occur: (1) the activity which determined the length of project delay will become a bounding activity; or, (2) the activity which determined the length of project delay will become continuous, but will not be a bounding activity. If event (2) above occurs, the bounding activities will remain the same, as will the bounded

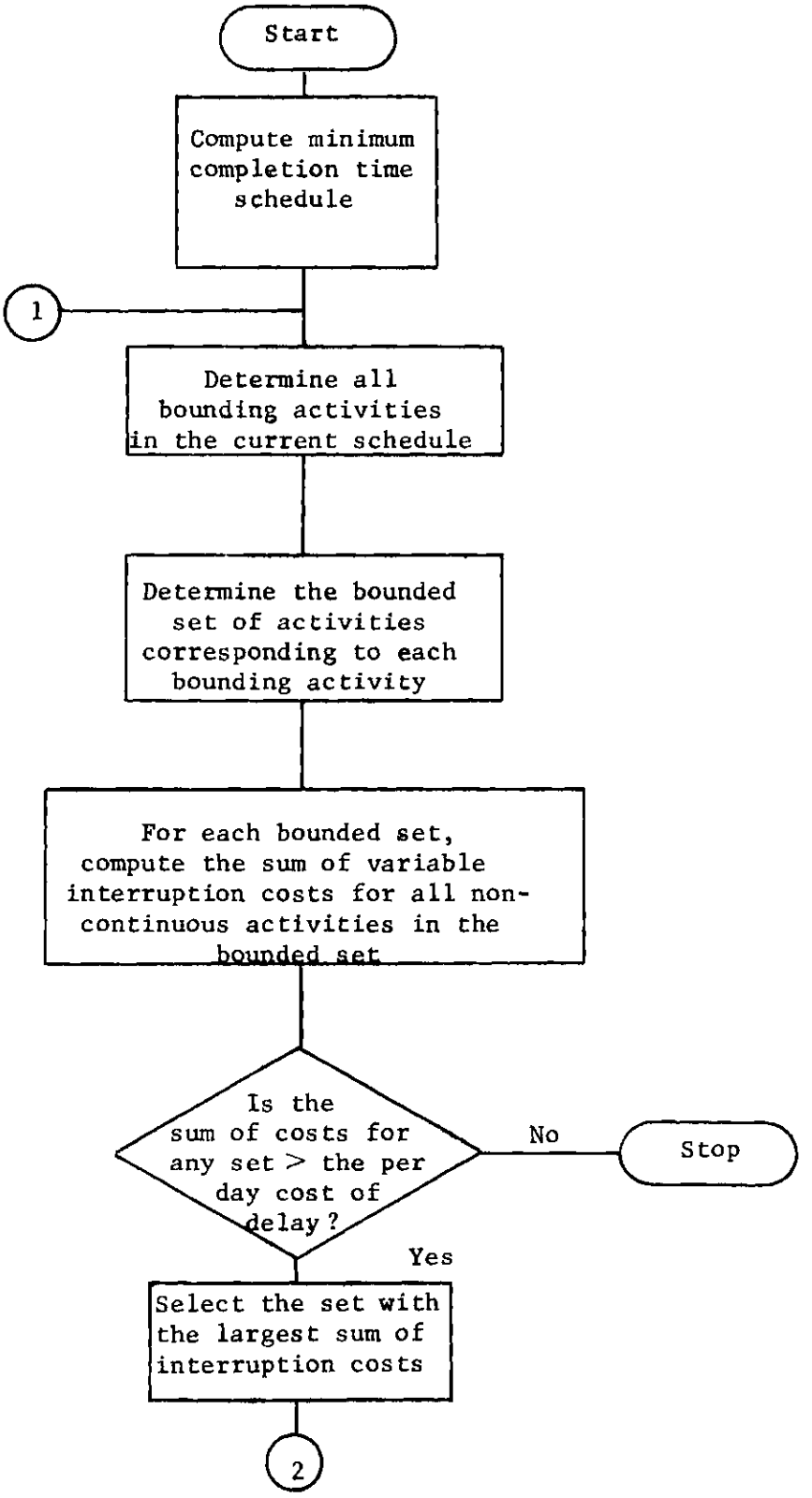


Figure 19. The Time-Cost Trade-Off Procedure

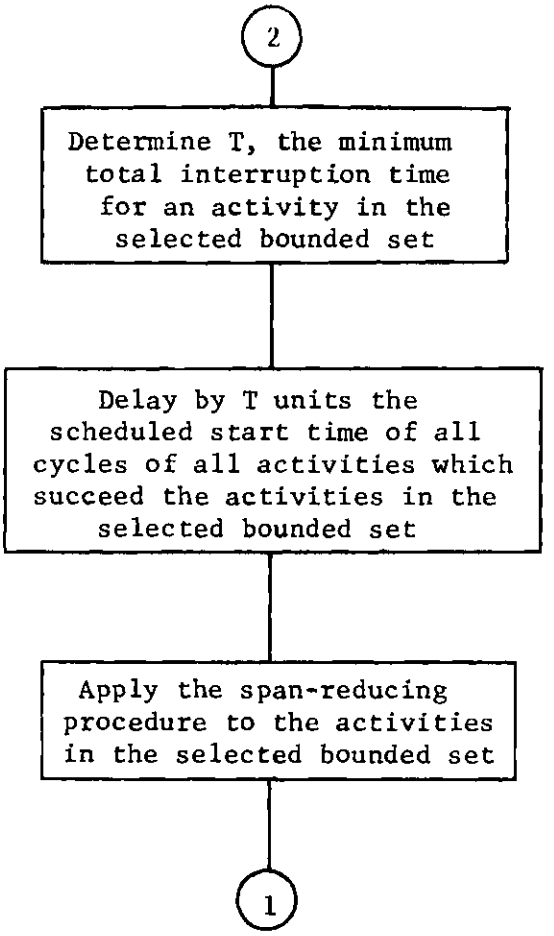


Figure 19. (Continued)

sets; only the contribution value of the span-reduced bounded set will be changed. However, if an activity which becomes continuous also becomes a bounding activity, then the bounded set will be sub-divided and it will be necessary to reconsider the bounding activities and bounded sets by returning to step 2.

In either case it is important to note that by selecting a project delay time such that at every iteration of the procedure an activity becomes continuous, the value of the bounded set with the largest contribution value at the next iteration of the procedure must be less than or equal to the largest contributed value at the current iteration. Thus at every iteration of the procedure, the largest contribution value among all the bounded sets will be less than or equal to the largest contributed value at the previous iteration. Therefore, at every iteration, the potential for reducing variable interruption costs is decreasing, or, at least, not increasing. This will result in the steepest-descent approach that is desired, giving a convex cost curve.

Additionally, because the potential for reducing costs is never increasing, it will be possible to determine when the procedure should be terminated. As indicated in step 5 of the algorithm, if the largest contribution value among all bounded sets is not greater than the project delay cost, there is no advantage in increasing project duration. Since the contribution values of all bounded sets at this (and any additional) iteration will be less than or equal to the current largest value, the procedure should be terminated because the potential for any further reduction in total project costs has been eliminated.

As an example of the use of the procedure, it was applied to the schedule of the problem of Figure 3. The schedule for this problem was given in Figure 8. A table showing a summary of the cost reduction for each iteration of the procedure is given in Table 2. As shown in the table, the minimum cost achieved for this project using the time-cost trade-off procedure was 310 and was obtained at a completion time of 139. From Figure 14, it is seen that the lowest cost attained by applying the modified cyclic project scheduling procedure over the acceptable lengths of the project yielded a lowest cost of 366 at a completion time of 149. Thus, the time-cost trade-off procedure has resulted in a significantly lower cost at an earlier completion time.

The computation details of the algorithm as applied to the example of Figure 3 are given in Appendix A. For each iteration, the bounding activities in the network are indicated, the bounded sets and their contribution values are presented, and the schedule for the project is given in tabular and Gantt chart form. The cost curve given by Figure 36 in Appendix A also demonstrates the resulting cost curve which is a close approximation to that of Figure 15.

Table 2. Summary of the Application of the Time-Cost Trade-Off Procedure to the Schedule of Figure 8

Cost of Delay (C_d) = 8

Iteration	Maximum Value of a Bounded Set	Delay Time	Cost Reduction	Costs			Completion Time
				Inter-ruption	Delay	Total	
1	37	2	74	384	16	400	131
2	22	6	132	252	64	316	137
3	11	2	22	230	80	310	139
4	5	2	10	220	96	316	141
5	4	46	184	36	464	500	187
6	3	8	24	12	528	540	195
7	2	6	12	0	576	576	201

CHAPTER VI

COMPUTER PROGRAMS FOR CYCLIC PROJECT SCHEDULING

In the preceding chapters, an improved version of the cyclic project scheduling algorithm of Ferraz (5) was developed and extended to other applications in the scheduling of cyclic projects. A steepest-descent time-cost trade-off procedure was also developed which enables the computation of a lower total cost based on the extension of the minimum completion time schedule. In order to demonstrate the applicability, usefulness, and efficiency of some of these procedures, computer programs have been developed to perform the scheduling computations of the modified scheduling algorithm.

The programs which have been developed were coded in the FORTRAN V language for use with the EXEC 8 operating system of the UNIVAC 1108 at the Georgia Institute of Technology. There are two programs: one for batch operation using card input and line printer output; the other for use in demand mode, with interactive features that allow decisions to be made at the terminal by the user in determining project schedules.

The Batch Mode Program

The first program developed was the batch mode program. The program accepts card input which contains information on the activities, their duration, and associated costs and other information. The program consists of a main segment and twenty subroutines, which perform the actual scheduling computations, provide tracing and debugging facilities, and

write formatted output. The program has been named "CPSS" which is the acronym for "Cyclic Project Scheduling System."

The program has been developed to have a capacity of fifty cycles of a fifty-node, 100 activity network. This is the equivalent of a project with 5000 activities and approximately 2500 nodes. All information on the activities is stored in arrays defined in the main program and no facilities have been provided for maintaining information on mass storage devices.

The program has been thoroughly debugged with the aid of a subroutine called RANGEN. This subroutine, using a random number generator, was used to provide random networks for scheduling by the program. A random number of cycles, nodes, activities, activity durations, and project costs was generated and provided as input to the scheduling subroutines of the program. For all networks generated, the program provided schedules without difficulty. The largest network generated consisted of thirty-four nodes, ninety-two activities, and seventeen cycles. This is equivalent to a project with 1554 activities and approximately 578 nodes. To compute and print a listing and Gantt chart of the schedule required less than twenty-five seconds of computer processing time. This gives some indication of the efficiency of the program. Most smaller networks were completely scheduled in fifteen seconds or less.

A more detailed description of the program is given in Appendix B. This section includes a macro flowchart of the execution of the program, as well as a variable listing, source listing, and description of input and output formats. A sample batch run, showing input and output for the problem of Figure 3, is also given.

The Demand Mode Program

Another program was also developed to allow more flexibility and interaction by the decision maker in the study of projects where various activities have been forced to be continuous or the project completion time has been extended. This program is called "CPSS-D," with the "D" indicating demand mode.

The demand mode program utilizes most of the same subroutines as the batch program for scheduling projects. The demand program CPSS-D, however, uses a slightly different main program to control the real-time execution of the scheduling subroutines. This main program directs inquiries to the user and, based upon the response, carries out the desired scheduling options. A brief sample of a demand mode interactive session is given in Figure 20.

The user first responds to inquiries concerning the number of activities, the number of cycles, the delay cost, and the selection of an appropriate objective function. The program then receives all of the input data for the activities of the program. The program then determines the minimum completion time schedule and prints a listing of the resulting costs. The maximum completion time schedule (the schedule with all activities continuous) is also determined and the maximum completion time is printed.

The user may then select particular activities to be continuous or select alternate completion times which are within the bounds of the minimum and maximum. When one of these options is selected, the appropriate schedule is computed and the resulting costs are printed. After the completion of any schedule, the user may inquire into the detailed schedule

```
 THE MINIMUM CØMPLETIØN TIME FØR THIS PRØJECT IS:      129

 THE TØTAL CØSTS ØF THIS SCHEDULE:
  VARIABLE INTERRUPTIØN CØST..    458
  FIXED INTERRUPTIØN CØST.....     58
  DELAY CØST..................      0
                                ------
            TØTAL CØST.........    516

 WØULD YØU LIKE ANY INFØ ØN THE
 SCHEDULED ACTIVITIES (YES/NØ)
YES

 YØU HAVE THE FØLLØWING ØPTIØNS:
      (1) LIST CØNTINUØUS ACTIVITIES
      (2) LIST NØN-CØNTINUØUS ACTIVITIES
      (3) LIST INDIVIDUAL ACTIVITIES
      (4) CØNTINUE SCHEDULING.
 ENTER 1,2,3,ØR 4
1
 CØNTINUØUS ACTIVITIES ARE
   I   J S(N,1)
   1   2      0
   3   4      6
   5   7     79
   5  10     40
   5  11     79
   5  12     45
   5  14     24
   7   9     80
   8  14     71
  11  15     82
  22  23    109

 YØU HAVE THE FØLLØWING ØPTIØNS:
      (1) LIST CØNTINUØUS ACTIVITIES
      (2) LIST NØN-CØNTINUØUS ACTIVITIES
      (3) LIST INDIVIDUAL ACTIVITIES
      (4) CØNTINUE SCHEDULING.
 ENTER 1,2,3,ØR 4
4

 THE MAXIMUM CØMPLETIØN TIME IS    201

 YØU NØW HAVE THE FØLLØWING ØPTIØNS:
      (1) SELECT ACTIVITIES TØ BE CØNTINUØUS
      (2) VARY THE PRØJECT CØMPLETIØN TIME
      (3) STØP
 ENTER 1, 2, ØR 3
```

Figure 20. Example of a Portion of an Interactive Demand Mode Scheduling Session

of any activity. The program may be terminated at any time the user desires to stop.

The advantage of this type of interactive program is that it allows very rapid computation of the costs of any schedule for a project. The user may alternately select goals or constraints for different schedules and find a schedule that satisfies his cost or completion time limitations.

The details of the use of this program are given in Appendix C. This appendix includes a listing of the program MAIN-D, a macro flowchart of the program, and two sample runs based on the network of Figure 3.

CHAPTER VII

CONCLUSIONS AND RECOMMENDATIONS

Conclusions

This research has focused on the scheduling of parallel multi-cycle projects with the objective of reducing costs or penalties resulting from interruptions between cycles of activities. Based upon an original algorithm developed by Ferraz, several modifications and extensions of the procedure were developed. These included: (1) definition of alternate penalty measures and objective functions for scheduling projects; (2) development of a procedure to reduce the span of activities in a project schedule; (3) modification of the scheduling procedure to allow selected activities to be continuous; and (4) modification of the procedure to allow a schedule to be computed with selected feasible completion times. Some of these procedures were developed as the result of an assumption that it was possible to allow the extension of the project completion date based upon some known costs.

An additional development of this research has been the formulation of a time-cost trade-off procedure which allows the computation of a low cost schedule when the daily cost of extending the project completion date is known. While similar to many time-cost trade-off procedures developed for non-cyclic critical path schedules, this method is different in that it focuses on the extension of project completion time in order to reduce the cost of activity interruptions.

Computer programs have also been developed which demonstrate some methods by which the scheduling algorithms could be adapted for use in computerized scheduling systems. These programs may serve as the prototype for a more sophisticated scheduling system developed specifically for cyclic projects.

Recommendations for Additional Research

This research represents some of the more advanced ideas on cyclic project scheduling. A survey of the literature on this subject has shown that little work has been done on this topic and that there is considerable need for additional research.

While having developed some significant heuristic algorithms for scheduling cyclic projects and reducing their costs, the research to date has not greatly considered the influence of resources on cyclic project scheduling, with the exception of the work by Burney (4). This represents perhaps the most significant additional area for research. Techniques must be developed which schedule cyclic projects under conditions of limited resources, as well as considering leveling applied resources.

Another important area will be in the programming of the time-cost trade-off procedure. This development has not been included in the existing version of the scheduling program. Considerable difficulty is foreseen in trying to develop efficient programs to trace paths through large networks in order to locate the bounding activities and bounded sets. However, the development of a computer program will be necessary if the time-cost trade-off procedure is to be of benefit to industry in the efficient scheduling of cyclic projects.

Other developments should focus on adapting the scheduling procedures to practical applications in industry. Sophisticated scheduling systems for computer implementation must be developed to perform the complex calculations of the algorithms. These would have to be similar to many of the project scheduling packages currently available commercially. When such facilities are available, studies should be made to determine the effectiveness of the procedures and the necessary modification which will allow them to take an important role in project scheduling.

APPENDICES

APPENDIX A

AN EXAMPLE OF THE APPLICATION OF THE TIME-COST TRADE-OFF PROCEDURE

This appendix gives the details of the application of the time-cost trade-off procedure as applied to the schedule of Figure 8. This is the minimum completion time schedule for the project network of Figure 3.

Table 3 gives a summary of the seven iterations of the procedure and the bounding activities, bounded sets, cost contributions, and delay time for each. The selected bounded set for each iteration is indicated by an asterisk (*) at the end of the list of the activities in the set.

The remainder of the tables and charts of the appendix are arranged to give an iteration-by-iteration presentation of the changes in the schedule as the project completion time is extended. For each iteration, there is: a network showing the bounding activities for that iteration; a table giving the schedule of the activities and the resulting interruptions and costs after rescheduling the activities; and a Gantt chart of the schedule at the end of the rescheduling. The final chart, Figure 36, shows the resulting cost curve. The lowest-cost schedule is found to result at a completion time of 139 and to give a cost of 310.

Table 3. Detailed Summary of the Steps in the Application of the Time-Cost Trade-Off Procedure to the Schedule of Figure 8 [Cost of delay (C_d) = 8]

Iteration Number	Bounding Activities	Bounded Set	Sum of Costs	Length of Delay
1	1-2	---	0	0
	3-4	2-3	3	8
	5-14	4-5	4	46
	22-23*	5-6, 5-7, 5-10, 5-11, 5-12, 6-8, 7-9, 8-10, 8-14, 10-12, 11-15, 12-13, 13-14, 14-15, 15-16, 15-17, 16-19, 17-19, 18-19, 19-21, 20-21, 21-22	37	2
2	1-2	---	0	0
	3-4	2-3	3	8
	5-7, 5-11, 5-14, 5-14, 7-9, 11-15	4-5	4	46
	15-16, 17-19, 18-19*	5-6, 5-10, 5-12, 6-8, 8-10, 8-14, 10-12, 12-13, 13-14, 14-15, 15-17	22	6
	19-21, 20-21	16-19	2	6
	22-23	21-22	5	2
3	1-2	---	0	0
	3-4	2-3	3	8
	5-6, 5-7, 5-10, 5-11, 5-12, 5-14, 6-8, 7-9, 8-10, 8-14, 10-12, 11-15	4-5	4	46
	15-16, 15-17, 17-19, 18-19*	12-13, 13-14, 14-15	11	2

Table 3. (Continued)

Iteration Number	Bounding Activities	Bounded Set	Sum of Costs	Length of Delay
3 (cont.)	19,21, 20-21	16-19	2	6
	22-23	21-22	5	2
4	1-2	---	0	0
	3-4	2-3	3	8
	5-6, 5-7, 5-10, 5-11, 5-12, 5-14, 6-8, 7-9, 8-10, 8-14, 10-12, 11-15, 12-13, 13-14, 12-13, 13-14, 14-15, 15-16, 15-17, 17-19, 18-19	4-5	4	46
	19-21, 20-21	16-19	2	6
	22-23*	21-22	5	2
5	1-2	---	0	0
	3-4	2-3	3	8
	5-6, 5-7, 5-10, 5-11, 5-12, 5-14, 6-8, 7-9, 8-10, 8-14, 10-12, 11-15, 12-13, 13-14, 14-15, 15-16, 15-17, 17-19, 18-19*	4-5	4	46
	19-21, 20-21, 21-22, 22-23	16-19	2	6
6	1-2	---	0	0
	3-4, 4-5, 5-6, 5-7, 5-10, 5-11, 5-12, 5-14, 6-8, 7-9, 8-10, 8-14,	2-3	3	8

Table 3. (Concluded)

Iteration Number	Bounding Activities	Bounded Set	Sum of Costs	Length of Delay
6 (cont.)	10-12, 11-15, 12-13, 13-14, 14-15, 15-16, 15-17, 17-19, 18-19*			
	19-21, 20-21, 21-22, 22-23	16-19	2	6
7	1-2, 2-3, 3-4, 4-5, 5-6, 5-7, 5-10, 5-11, 5-12, 5-14, 6-8, 7-9, 8-10, 8-14, 10-12, 11-15, 12-13, 13-14, 14-15, 15-16, 15-17, 17-19, 18-19	---	0	0
	19-21, 20-21, 21-22, 22-23*	16-19	2	6

Table 4. Initial Activity Schedules and Costs

Activity	Schedule					Interruptions		Cost/Unit (V_i)	Total Cost
	1	2	3	4	5	Number	Length		
1 - 2	0	4	8	12	16	0	0	2	0
2 - 3	4	14	16	18	20	1	8	3	24
3 - 4	6	20	34	48	62	0	0	1	0
4 - 5	22	38	54	63	76	4	46	4	184
5 - 6	50	55	60	65	78	1	8	1	8
5 - 7	79	80	81	82	83	0	0	3	0
5 - 10	40	50	60	70	80	0	0	1	0
5 - 11	79	82	85	88	91	0	0	5	0
5 - 12	45	55	65	75	85	0	0	3	0
5 - 14	24	40	56	72	88	0	0	2	0
6 - 8	55	60	65	70	83	1	8	2	16
7 - 9	80	82	84	86	88	0	0	2	0
8 - 10	60	65	70	75	88	1	8	2	16
8 - 14	71	76	81	86	91	0	0	2	0
10 - 12	65	70	75	80	93	1	8	3	24
11 - 15	82	85	88	91	94	0	0	2	0
12 - 13	76	79	82	85	98	1	10	4	40
13 - 14	79	82	85	88	101	1	10	3	30
14 - 15	82	85	88	91	104	1	10	4	40
15 - 16	85	90	95	100	107	1	2	1	2
15 - 17	87	90	93	96	107	1	8	3	24
16 - 19	92	95	102	105	112	2	8	2	16
17 - 19	90	95	100	105	112	1	2	2	4
18 - 19	90	95	100	105	112	1	2	1	2
19 - 21	95	100	105	110	117	1	2	1	2
20 - 21	95	100	105	110	117	1	2	3	6
21 - 22	106	109	112	115	122	1	4	5	20
22 - 23	109	113	117	121	125	0	0	2	0

Total = 458

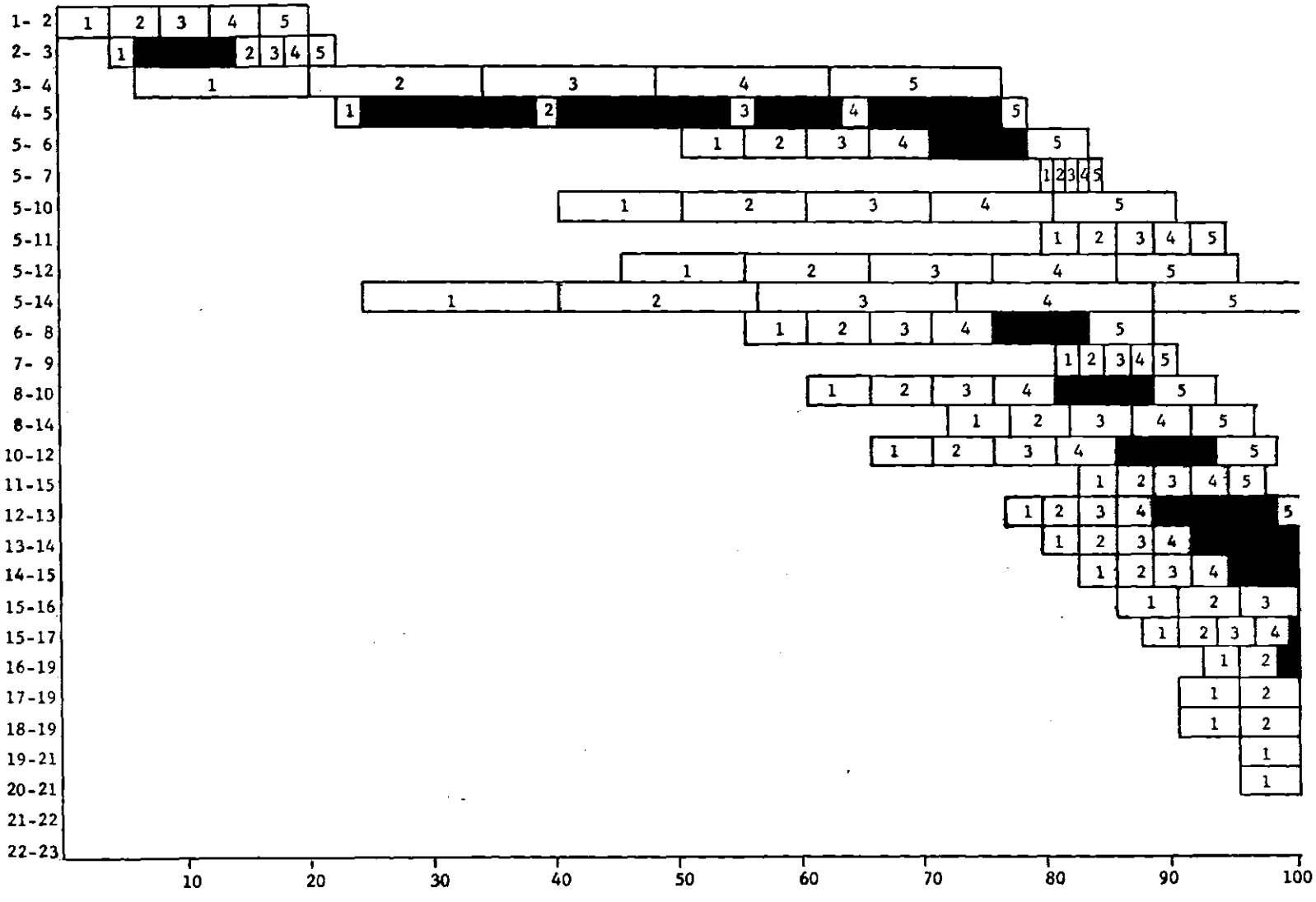


Figure 21. Gantt Chart of the Initial Schedule

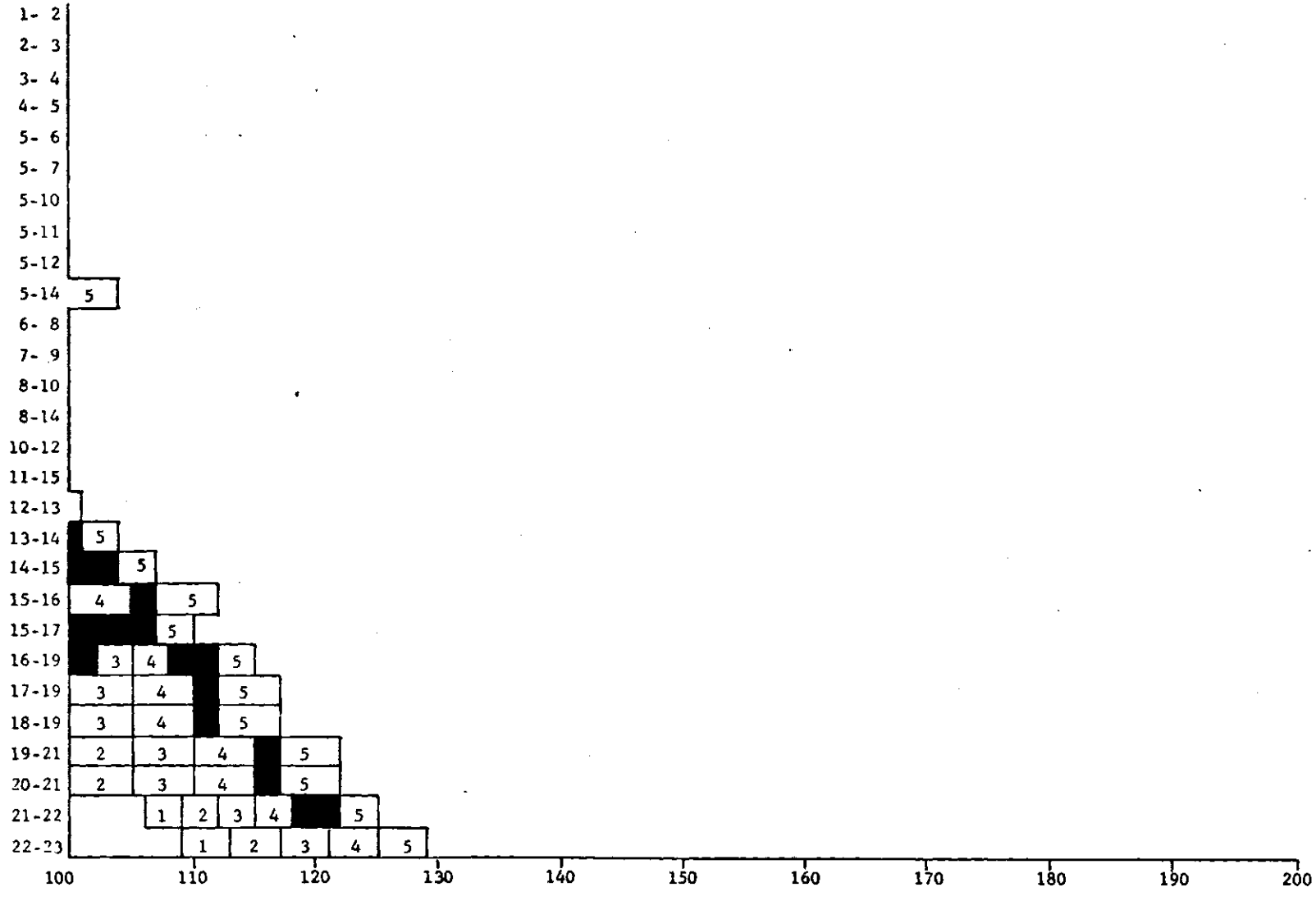


Figure 21. (Continued)

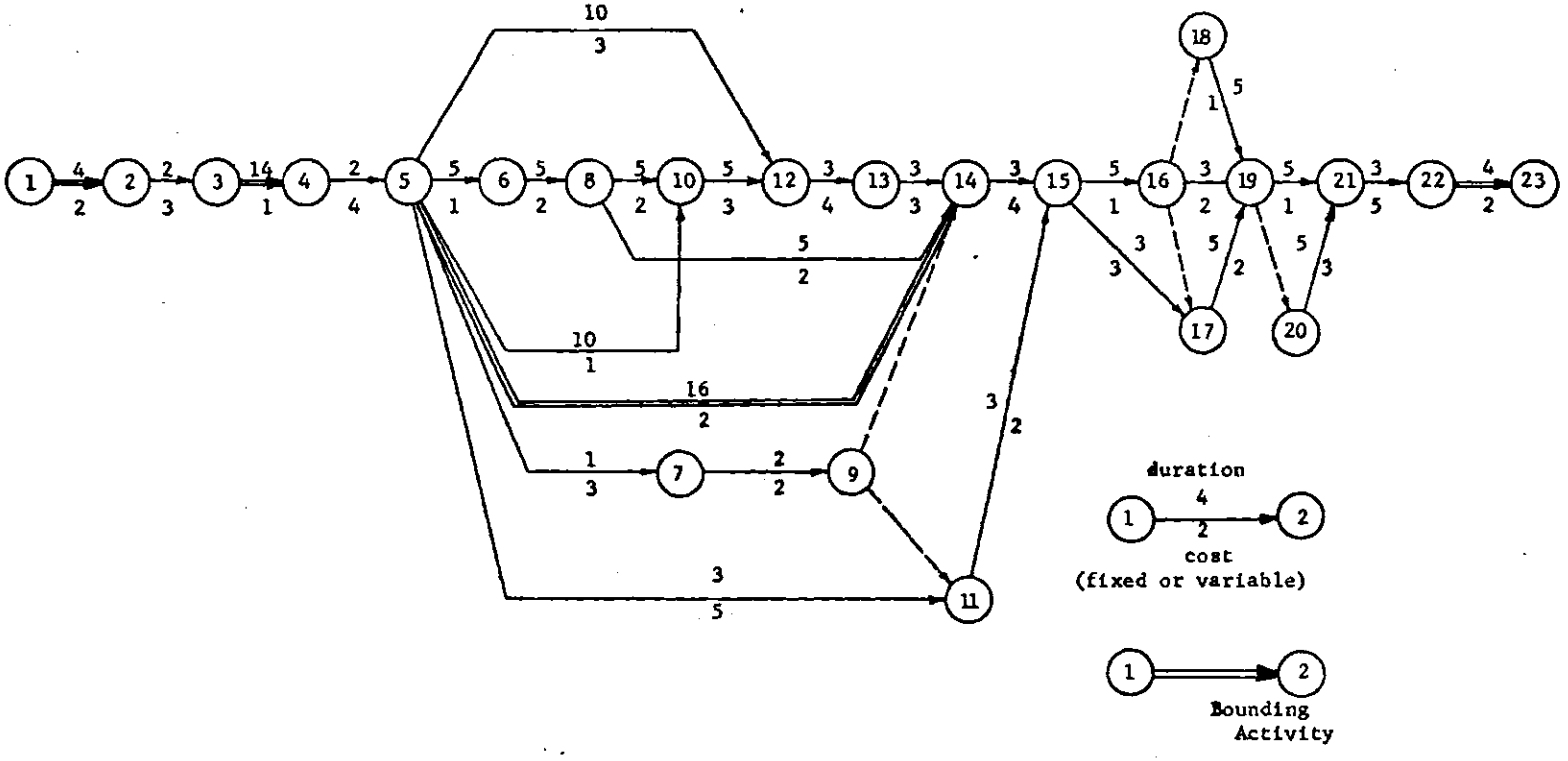


Figure 22. Bounding Activities for the First Iteration

Table 5. Activity Schedules and Costs at the End of the First Iteration

Activity	Schedule					Interruptions		Cost/Unit (V_i)	Total Cost
	1	2	3	4	5	Number	Length		
1 - 2	0	4	8	12	16	0	0	2	0
2 - 3	4	14	16	18	20	1	8	3	24
3 - 4	6	20	34	48	62	0	0	1	0
4 - 5	22	38	54	65	76	4	46	4	184
5 - 6	52	57	62	67	78	1	6	1	6
5 - 7	81	82	83	84	85	0	0	3	0
5 - 10	42	52	62	72	82	0	0	1	0
5 - 11	81	84	87	90	93	0	0	5	0
5 - 12	47	57	67	77	87	0	0	3	0
5 - 14	24	40	56	72	88	0	0	2	0
6 - 8	57	62	67	72	83	1	6	2	12
7 - 9	82	84	86	88	90	0	0	2	0
8 - 10	62	67	72	77	88	1	6	2	12
8 - 14	73	78	83	88	93	0	0	2	0
10 - 12	67	72	77	82	93	1	6	3	18
11 - 15	84	87	90	93	96	0	0	2	0
12 - 13	78	81	84	87	98	1	8	4	32
13 - 14	81	84	87	90	101	1	8	3	24
14 - 15	84	87	90	93	104	1	8	4	32
15 - 16	87	92	97	102	107	0	0	1	0
15 - 17	89	92	95	98	107	1	6	3	18
16 - 19	94	97	104	107	112	2	6	2	12
17 - 19	92	97	102	107	112	0	0	2	0
18 - 19	92	97	102	107	112	0	0	1	0
19 - 21	97	102	107	112	117	0	0	1	0
20 - 21	97	102	107	112	117	0	0	3	0
21 - 22	108	111	114	117	122	1	2	5	10
22 - 23	111	115	119	123	127	0	0	2	0

Total = 384

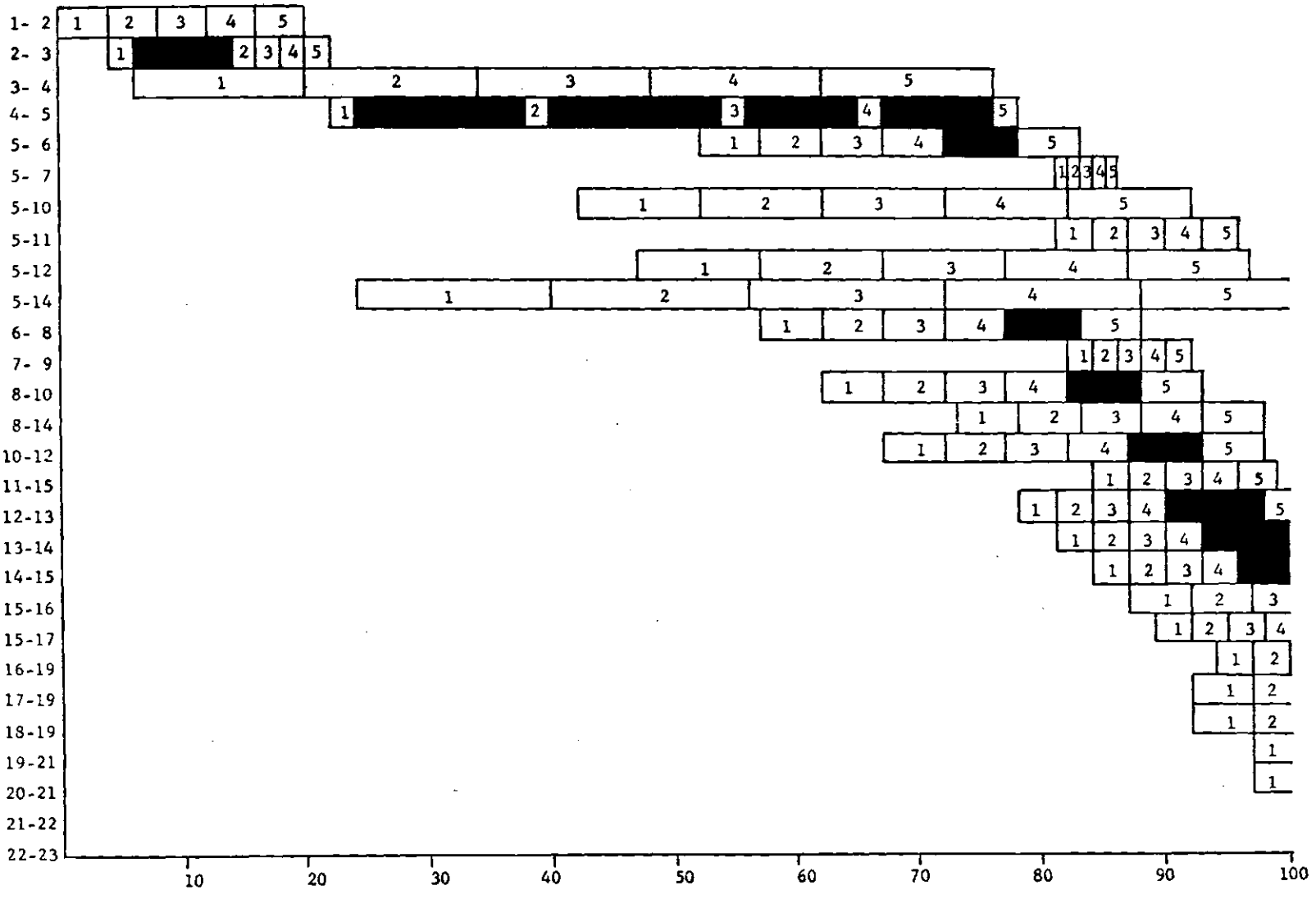


Figure 23. Gantt Chart After the First Iteration

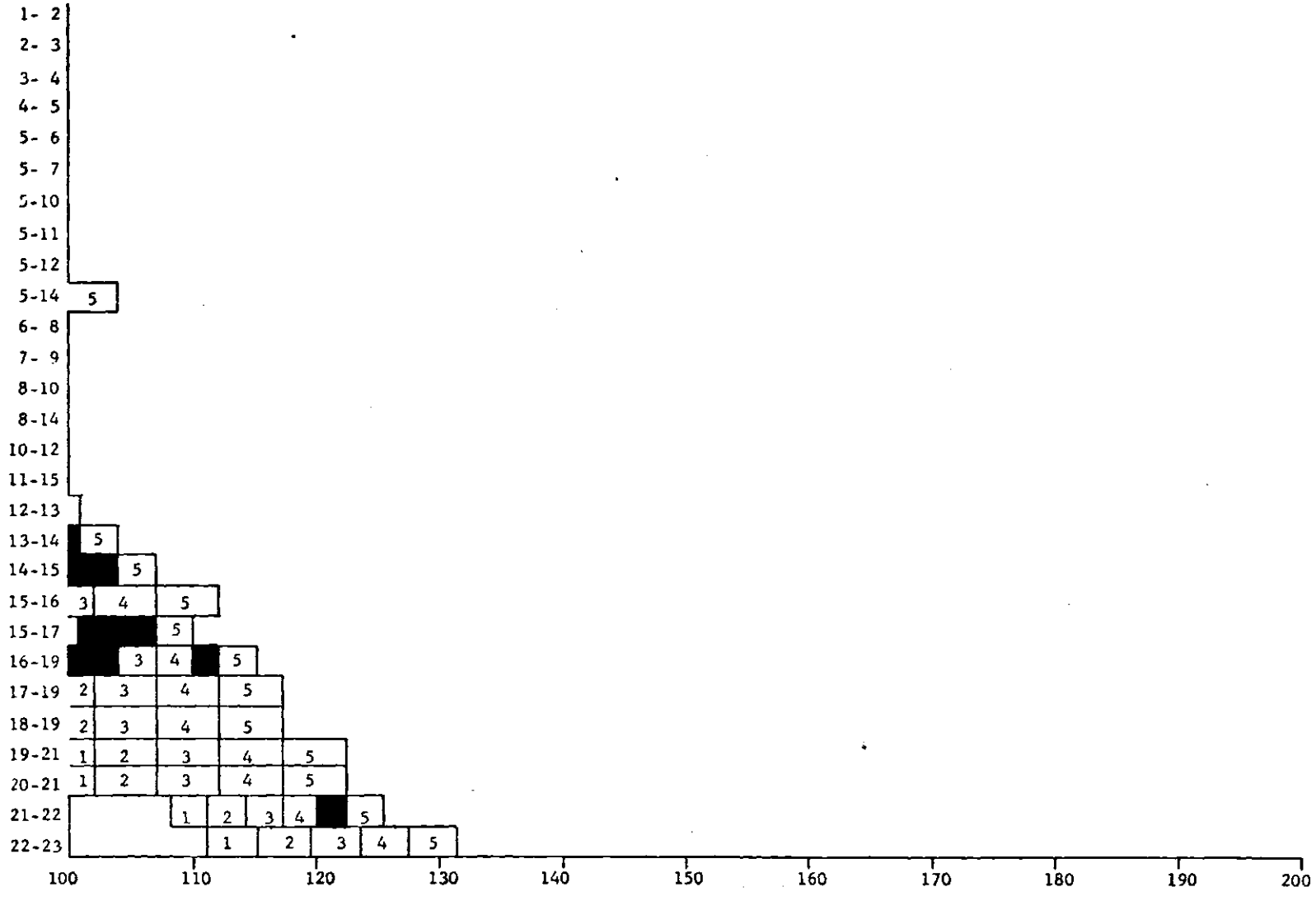


Figure 23. (Continued)

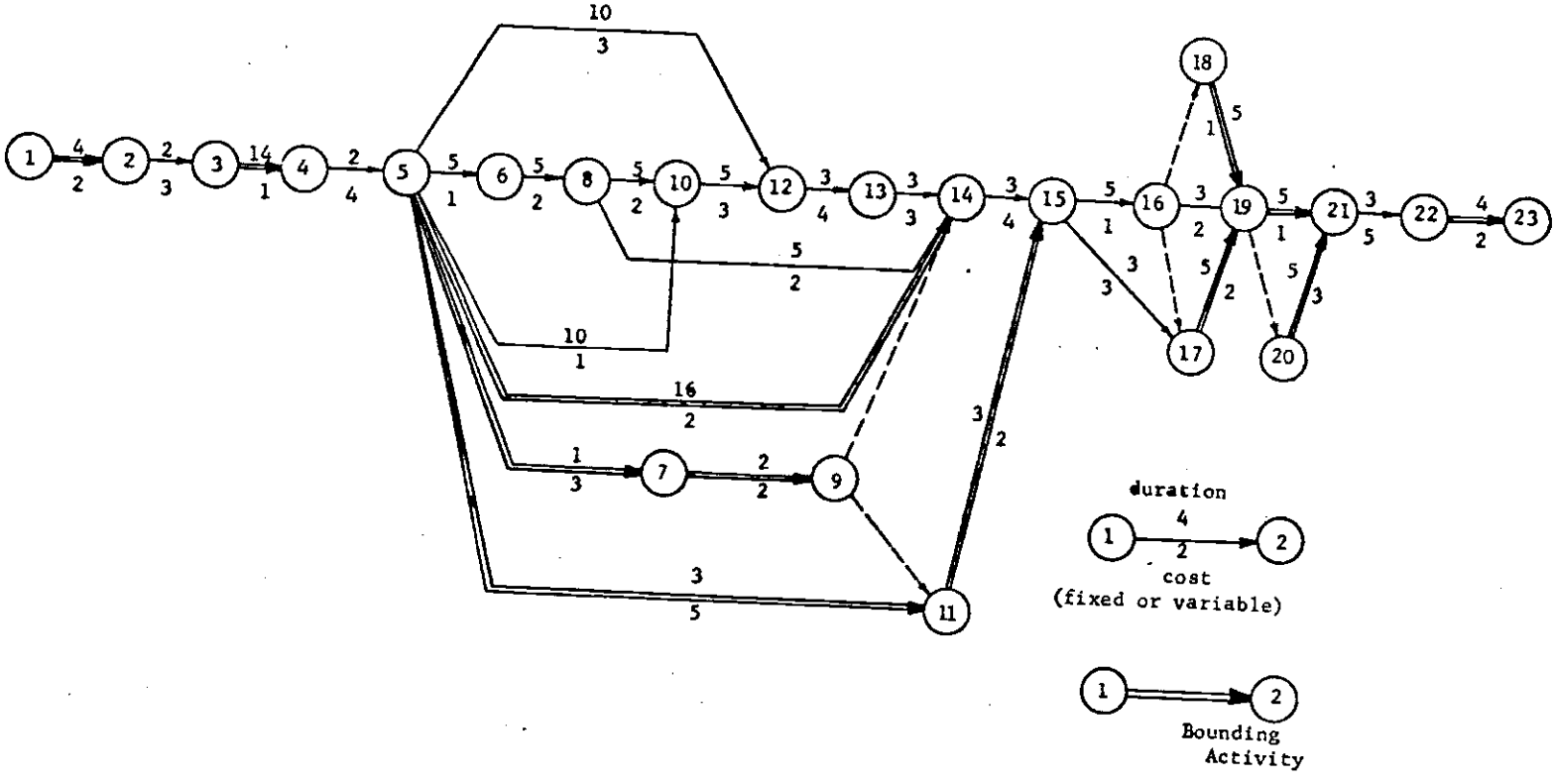


Figure 24. Bounding Activities for the Second Iteration

Table 6. Activity Schedules and Costs at the End of the Second Iteration

Activity	Schedule					Interruptions		Cost/Unit (V_i)	Total Cost
	1	2	3	4	5	Number	Length		
1 - 2	0	4	8	12	16	0	0	2	0
2 - 3	4	14	16	18	20	1	8	3	24
3 - 4	6	20	34	48	62	0	0	1	0
4 - 5	22	38	54	70	76	4	46	4	184
5 - 6	58	63	68	73	78	0	0	1	0
5 - 7	87	88	89	90	91	0	0	3	0
5 - 10	43	53	63	73	83	0	0	1	0
5 - 11	87	90	93	96	99	0	0	5	0
5 - 12	48	58	68	78	88	0	0	3	0
5 - 14	24	40	56	72	88	0	0	2	0
6 - 8	63	68	73	78	83	0	0	2	0
7 - 9	88	90	92	94	96	0	0	2	0
8 - 10	68	73	78	83	88	0	0	2	0
8 - 14	79	84	89	94	99	0	0	2	0
10 - 12	73	78	83	88	93	0	0	3	0
11 - 15	90	93	96	99	102	0	0	2	0
12 - 13	84	87	90	93	98	1	2	4	8
13 - 14	87	90	93	96	101	1	2	3	6
14 - 15	90	93	96	99	104	1	2	4	8
15 - 16	93	98	103	108	113	0	0	1	0
15 - 17	95	98	101	104	107	0	0	3	0
16 - 19	100	103	110	113	118	2	6	2	12
17 - 19	98	103	108	113	118	0	0	2	0
18 - 19	98	103	108	113	118	0	0	1	0
19 - 21	103	108	113	118	123	0	0	1	0
20 - 21	103	108	113	118	123	0	0	3	0
21 - 22	114	117	120	123	128	1	2	5	10
22 - 23	117	121	125	129	133	0	0	2	0

Total = 252

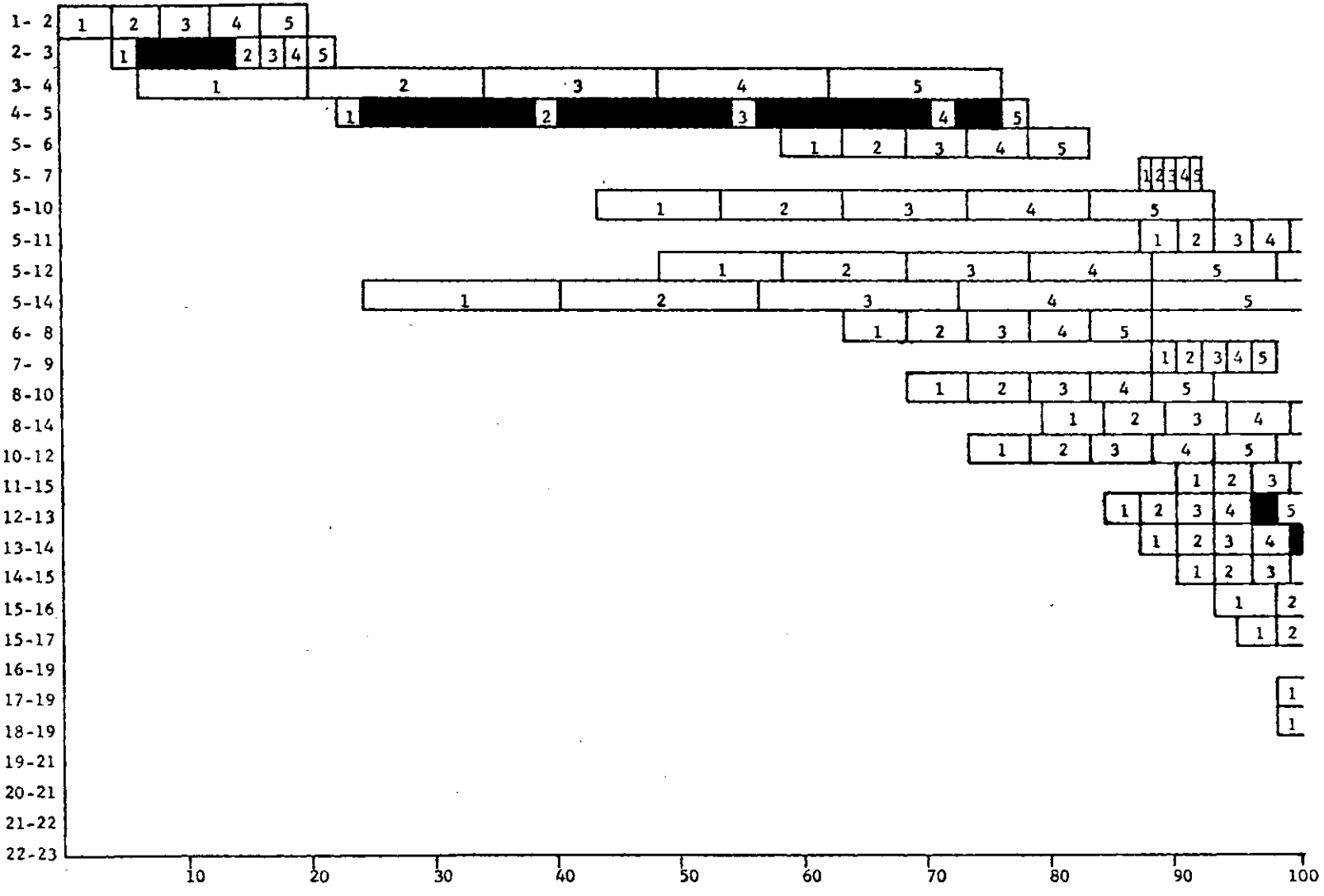


Figure 25. Gantt Chart After the Second Iteration

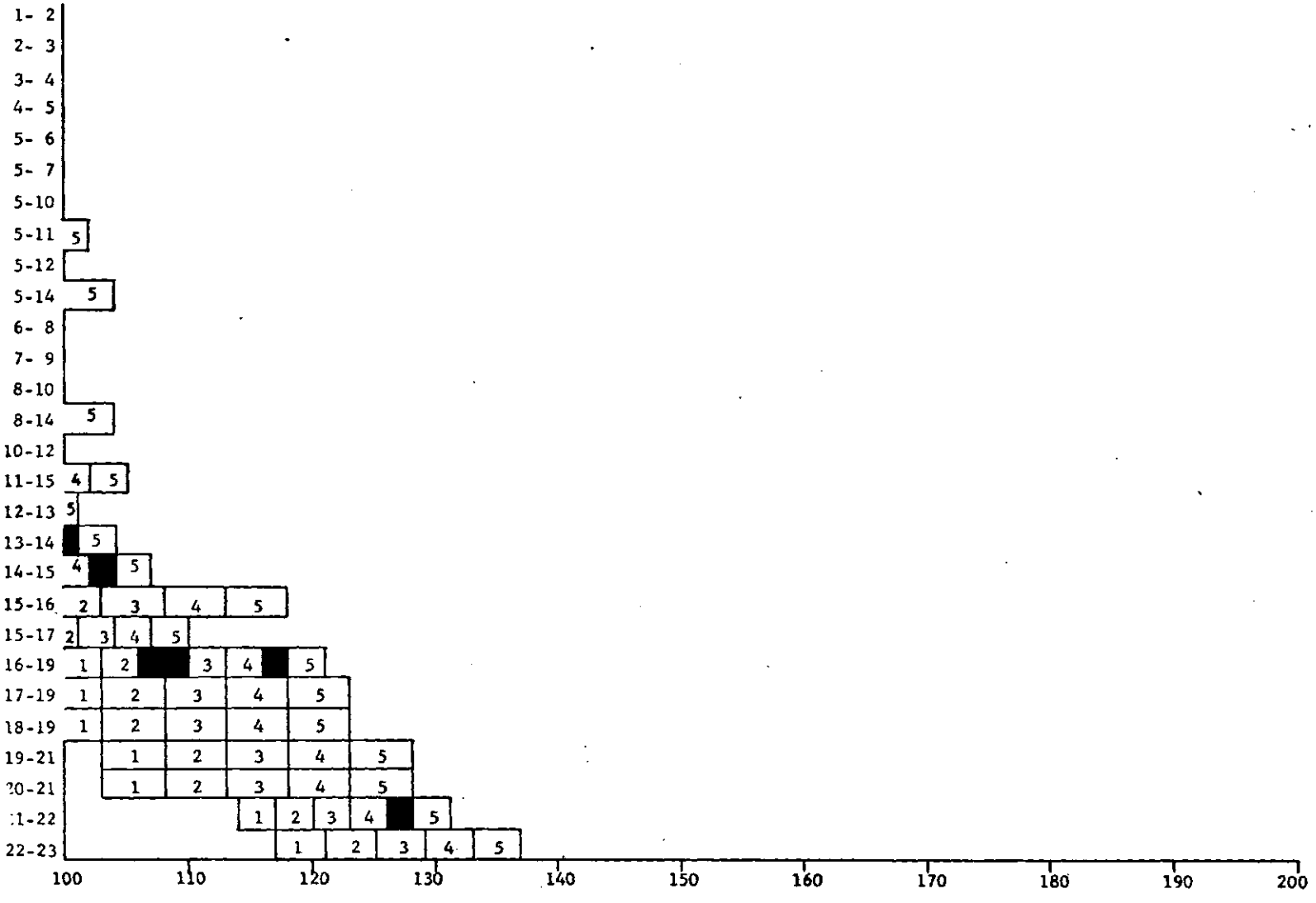


Figure 25. (Continued)

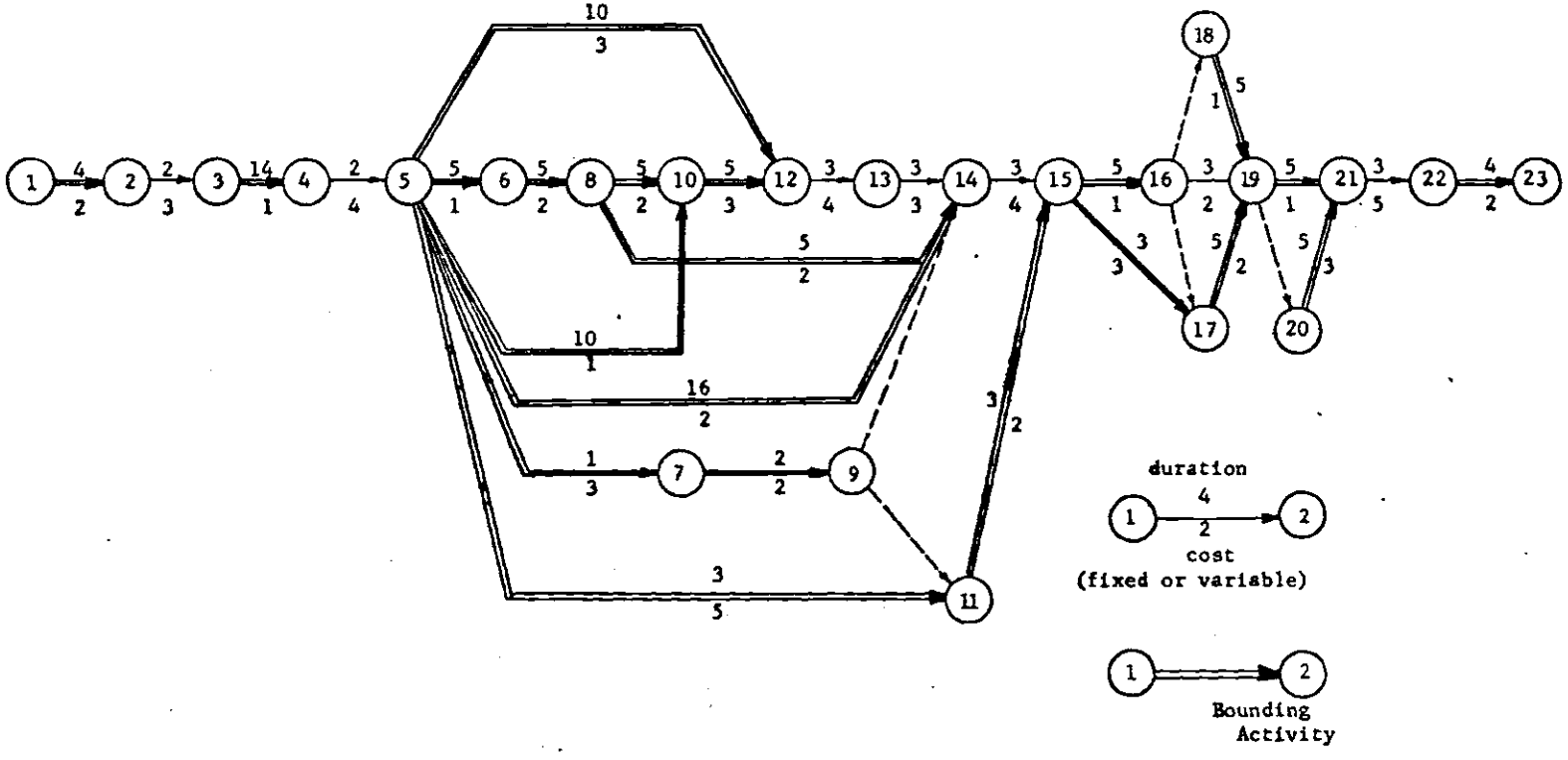


Figure 26. Bounding Activities for the Third Iteration

Table 7. Activity Schedules and Costs at the End of the Third Iteration

Activity	Schedule					Interruptions		Cost/Unit (V_i)	Total Cost
	1	2	3	4	5	Number	Length		
1 - 2	0	4	8	12	16	0	0	2	0
2 - 3	4	14	16	18	20	1	8	3	24
3 - 4	6	20	34	48	62	0	0	1	0
4 - 5	22	38	54	70	76	4	46	4	184
5 - 6	58	63	68	73	78	0	0	1	0
5 - 7	89	90	91	92	93	0	0	3	0
5 - 10	43	53	63	73	83	0	0	1	0
5 - 11	89	92	95	98	101	0	0	5	0
5 - 12	48	58	68	78	88	0	0	3	0
5 - 14	24	40	56	72	88	0	0	2	0
6 - 8	63	68	73	78	83	0	0	2	0
7 - 9	90	92	94	96	98	0	0	2	0
8 - 10	68	73	78	83	88	0	0	2	0
8 - 14	79	84	89	94	99	0	0	2	0
10 - 12	73	78	83	88	93	0	0	3	0
11 - 15	92	95	98	101	104	0	0	2	0
12 - 13	86	89	92	95	98	0	0	4	0
13 - 14	89	92	95	98	101	0	0	3	0
14 - 15	92	95	98	101	104	0	0	4	0
15 - 16	95	100	105	110	115	0	0	1	0
15 - 17	97	100	103	106	107	0	0	3	0
16 - 19	102	105	112	115	120	2	6	2	12
17 - 19	100	105	110	115	120	0	0	2	0
18 - 19	100	105	110	115	120	0	0	1	0
19 - 21	105	110	115	120	125	0	0	1	0
20 - 21	105	110	115	120	125	0	0	3	0
21 - 22	116	119	122	125	130	1	2	5	10
22 - 23	119	123	127	131	135	0	0	2	0

Total = 230

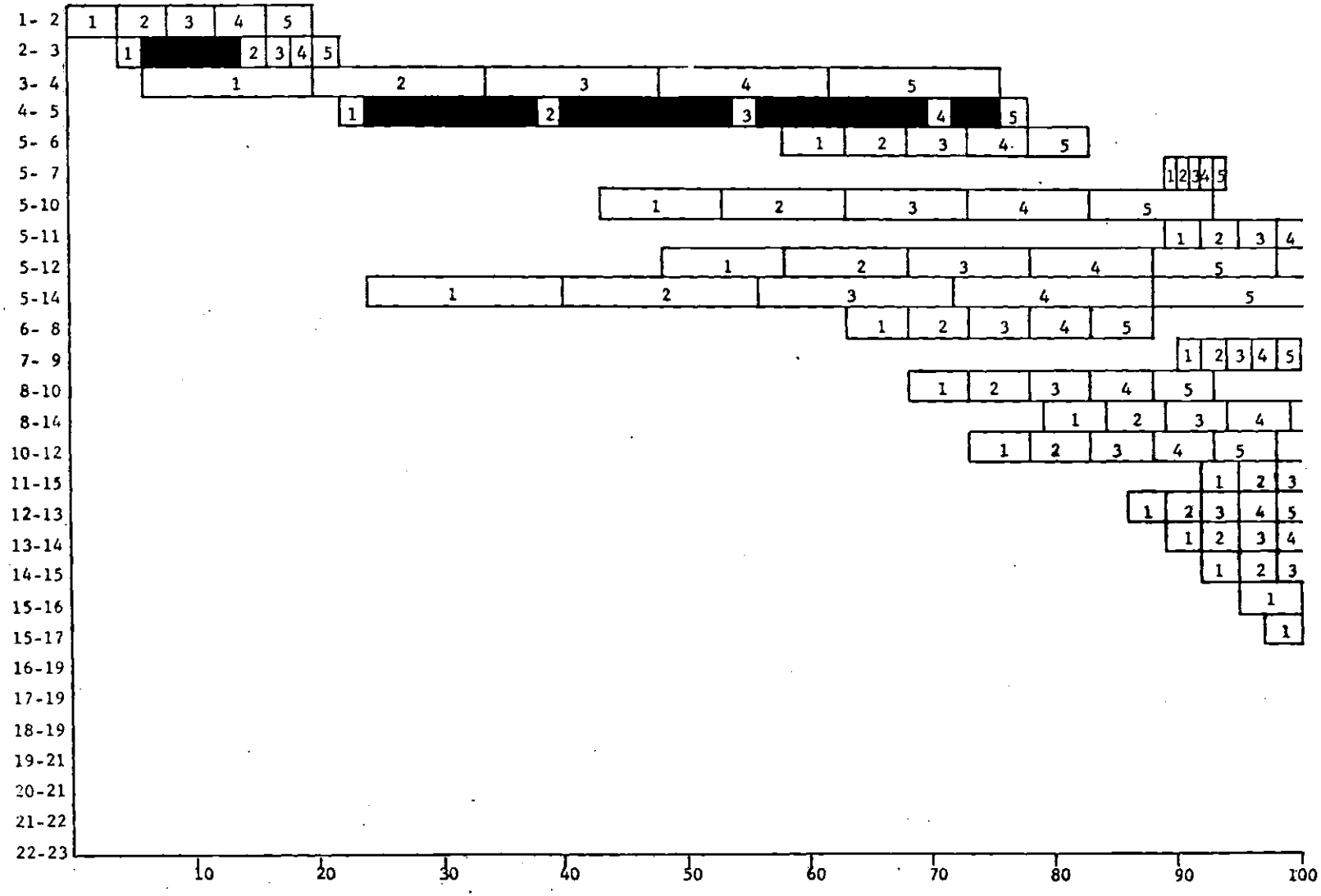


Figure 27. Gantt Chart After the Third Iteration

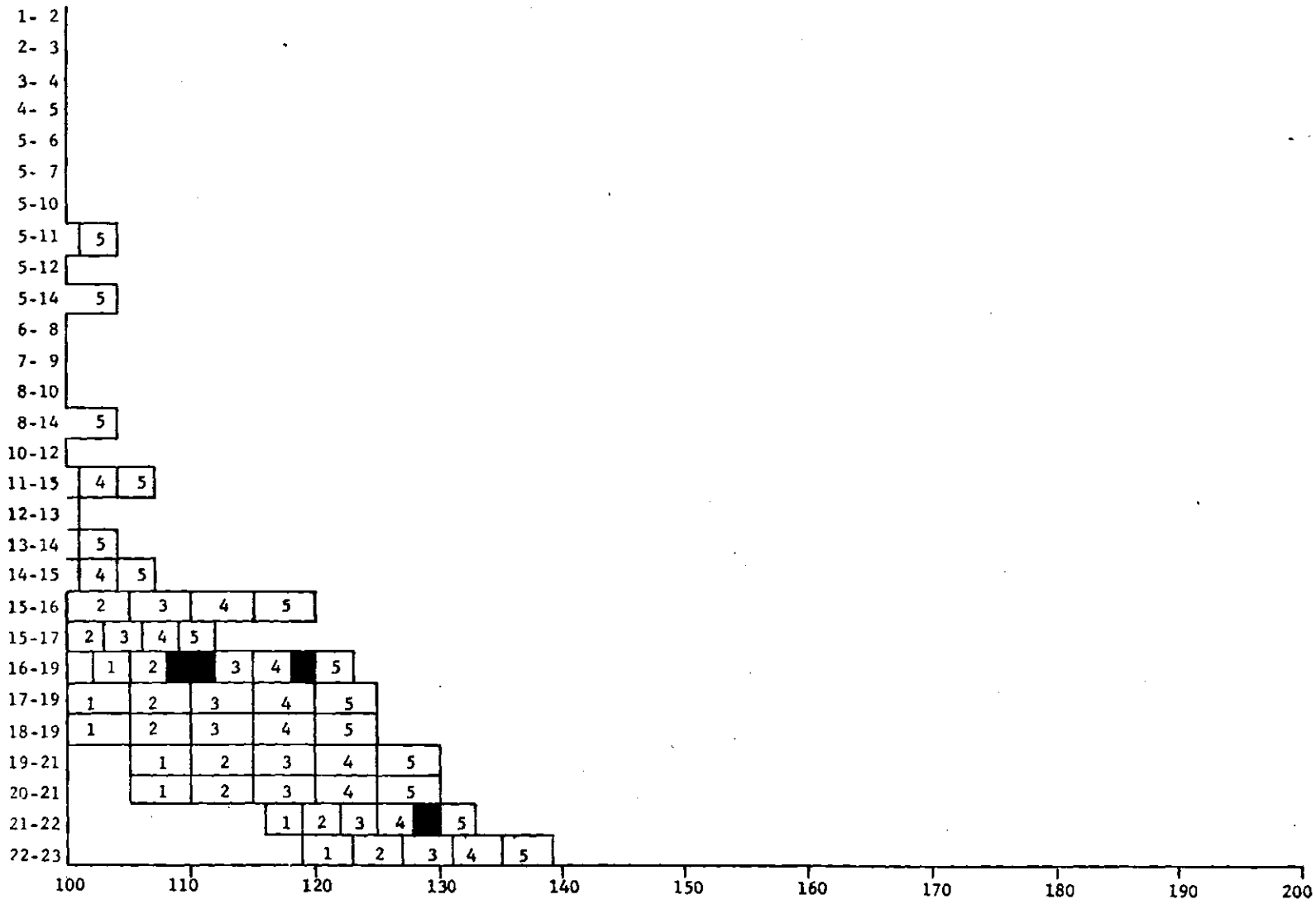


Figure 27. (Continued)

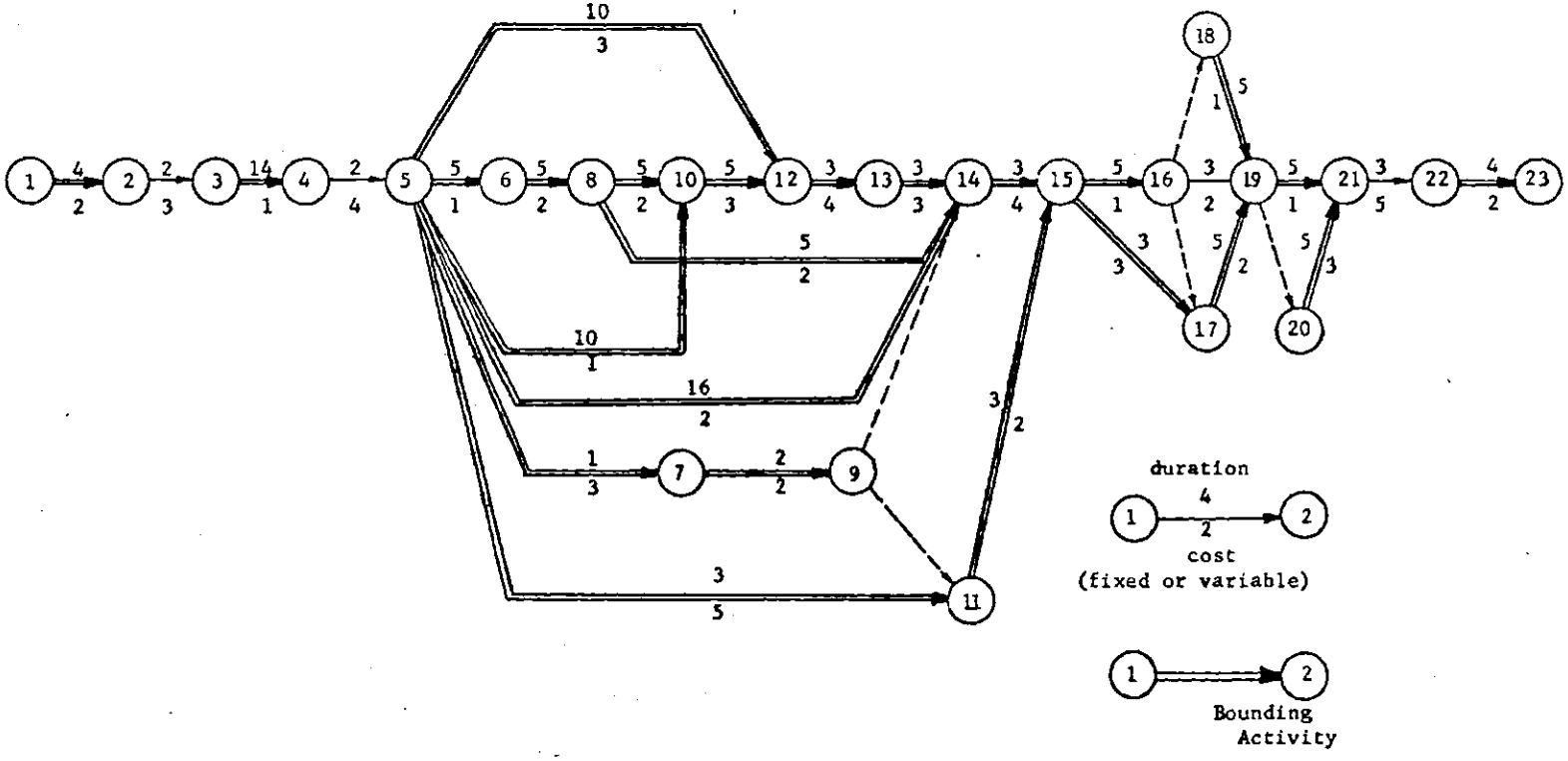


Figure 28. Bounding Activities for the Fourth Iteration

Table 8. Activity Schedules and Costs at the End of the Fourth Iteration

Activity	Schedule					Interruptions		Cost/Unit (V_i)	Total Cost
	1	2	3	4	5	Number	Length		
1 - 2	0	4	8	12	16	0	0	2	0
2 - 3	4	14	16	18	20	1	8	3	24
3 - 4	6	20	34	48	62	0	0	1	0
4 - 5	22	38	54	70	76	4	46	4	184
5 - 6	58	63	68	73	78	0	0	1	0
5 - 7	89	90	91	92	93	0	0	3	0
5 - 10	93	53	63	73	83	0	0	1	0
5 - 11	89	92	95	98	101	0	0	5	0
5 - 12	48	58	68	78	88	0	0	3	0
5 - 14	24	40	56	72	88	0	0	2	0
6 - 8	63	68	73	78	83	0	0	2	0
7 - 9	90	92	94	96	98	0	0	2	0
8 - 10	68	73	78	83	88	0	0	2	0
8 - 14	79	84	89	94	99	0	0	2	0
10 - 12	73	78	83	88	93	0	0	3	0
11 - 15	92	95	98	101	104	0	0	2	0
12 - 13	86	89	92	95	98	0	0	4	0
13 - 14	89	92	95	98	101	0	0	3	0
14 - 15	92	95	98	101	104	0	0	4	0
15 - 16	95	100	105	110	115	0	0	1	0
15 - 17	97	100	103	106	109	0	0	3	0
16 - 19	102	105	112	115	120	2	6	2	12
17 - 19	100	105	110	115	120	0	0	2	0
18 - 19	100	105	110	115	120	0	0	1	0
19 - 21	105	110	115	120	125	0	0	1	0
20 - 21	102	110	115	120	125	0	0	3	0
21 - 22	118	121	124	127	130	0	0	5	0
22 - 23	121	125	129	133	137	0	0	2	0

Total = 220

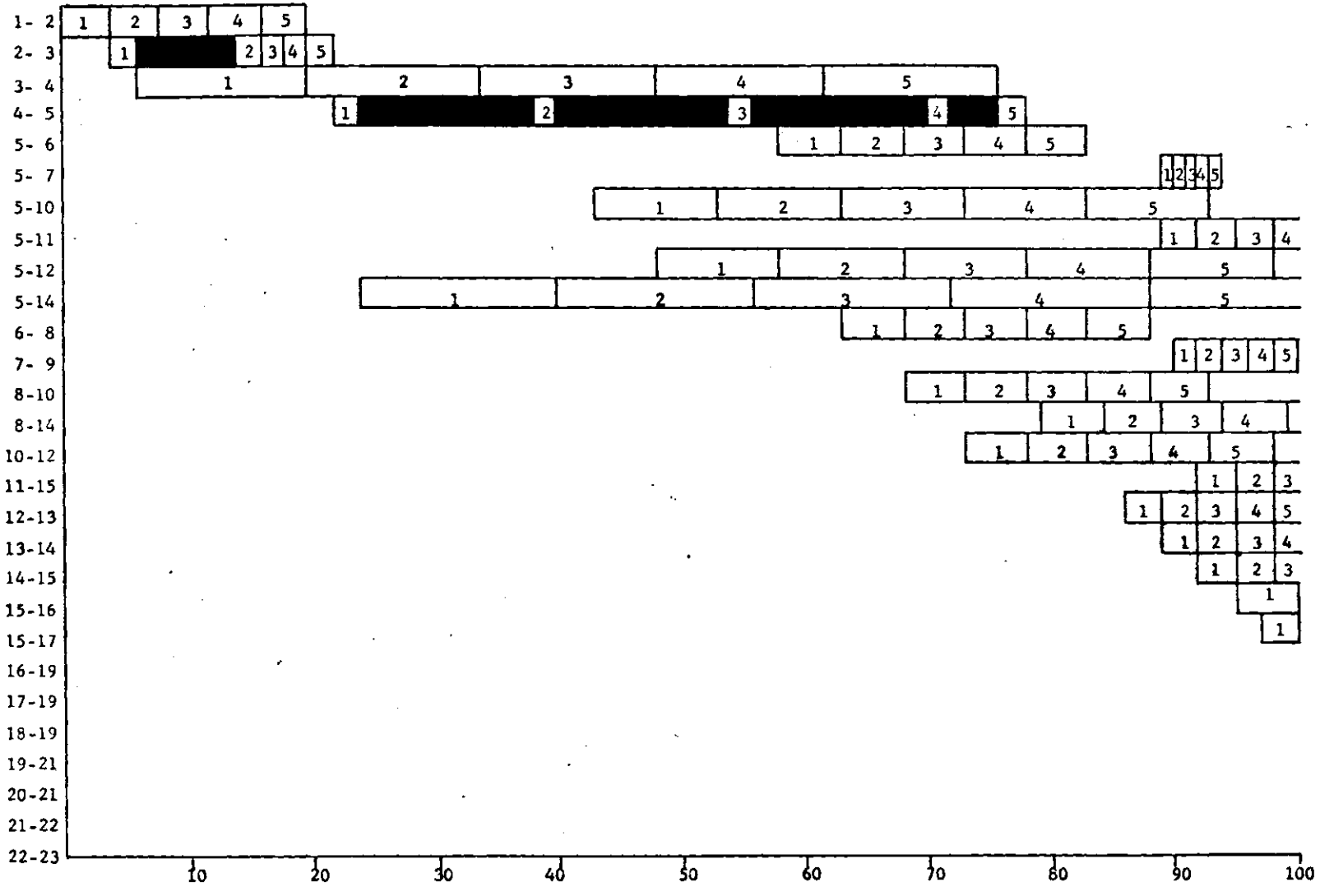


Figure 29. Gantt Chart After the Fourth Iteration

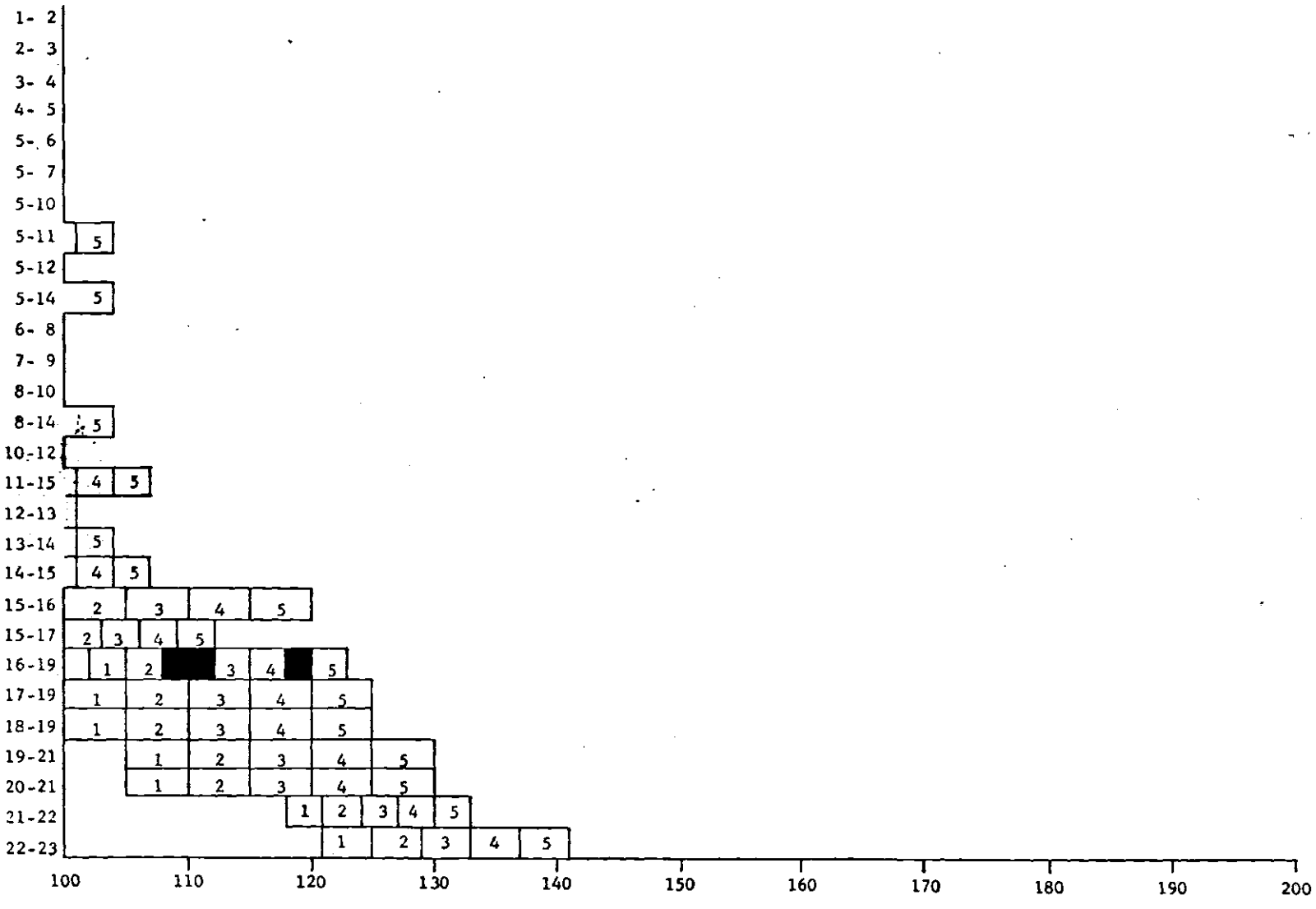


Figure 29. (Continued)

Table 9. Activity Schedules and Costs at the End of the Fifth Iteration

Activity	Schedule					Interruptions		Cost/Unit (V_i)	Total Cost
	1	2	3	4	5	Number	Length		
1 - 2	0	4	8	12	16	0	0	2	0
2 - 3	4	14	16	18	20	1	8	3	24
3 - 4	6	20	34	48	62	0	0	1	0
4 - 5	68	70	72	74	76	0	0	4	0
5 - 6	104	109	114	119	124	0	0	1	0
5 - 7	135	136	137	138	139	0	0	3	0
5 - 10	89	99	109	119	129	0	0	1	0
5 - 11	135	138	141	144	147	0	0	5	0
5 - 12	94	104	114	124	134	0	0	3	0
5 - 14	70	86	102	118	134	0	0	2	0
6 - 8	109	114	119	124	129	0	0	2	0
7 - 9	136	138	140	142	144	0	0	2	0
8 - 10	114	119	124	129	134	0	0	2	0
8 - 14	125	130	135	140	145	0	0	2	0
10 - 12	119	124	129	134	139	0	0	3	0
11 - 15	138	141	144	147	150	0	0	2	0
12 - 13	132	135	138	141	144	0	0	4	0
13 - 14	135	138	141	144	147	0	0	3	0
14 - 15	138	141	144	147	150	0	0	4	0
15 - 16	141	146	151	156	161	0	0	1	0
15 - 17	143	146	149	152	155	0	0	3	0
16 - 19	148	151	158	161	166	2	6	2	12
17 - 19	146	151	156	161	166	0	0	2	0
18 - 19	146	151	156	161	166	0	0	1	0
19 - 21	151	156	161	166	171	0	0	1	0
20 - 21	151	156	161	166	171	0	0	3	0
21 - 22	164	167	170	173	176	0	0	5	0
22 - 23	167	171	175	179	183	0	0	2	0

Total = 36

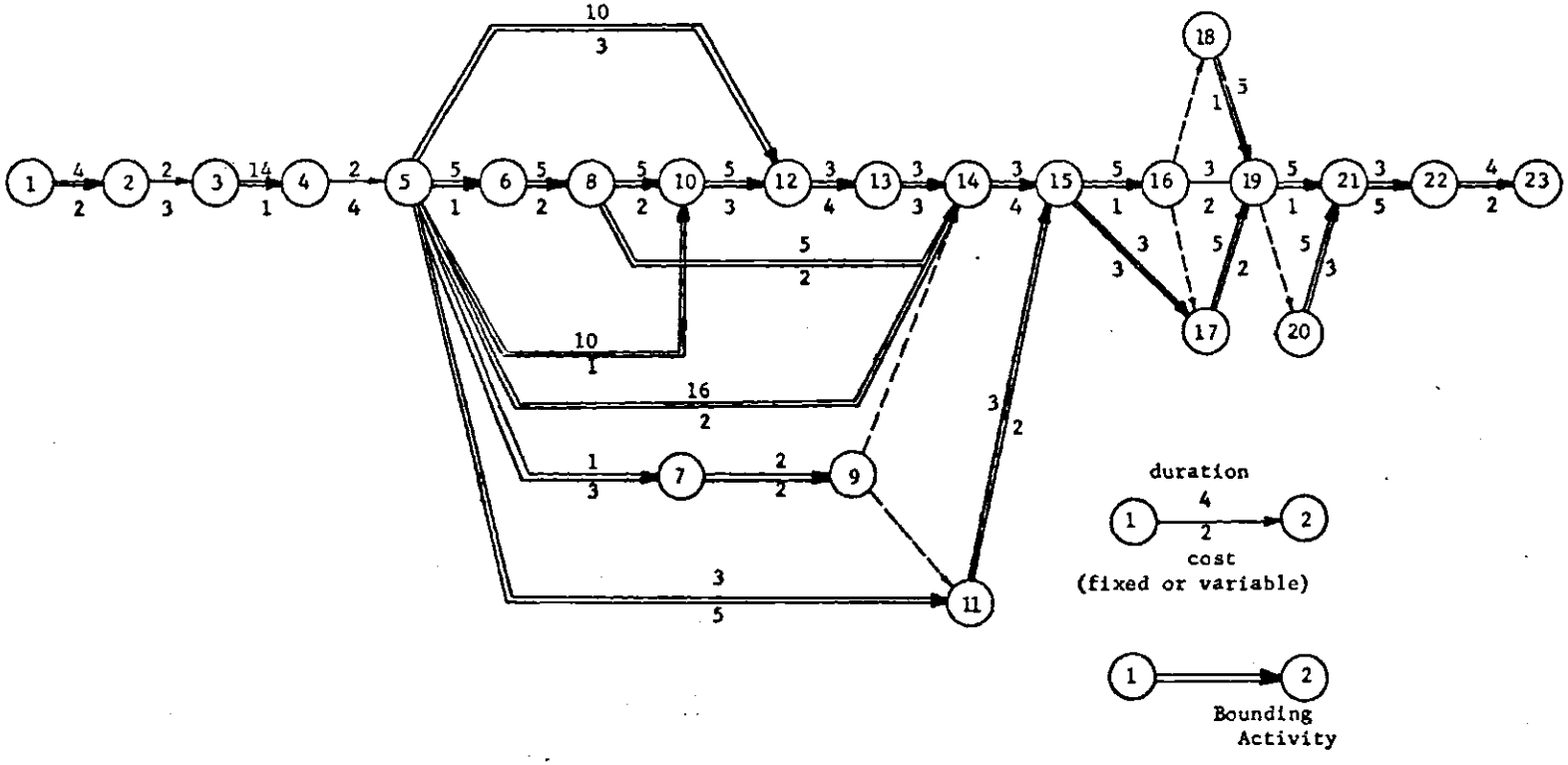


Figure 30. Bounding Activities for the Fifth Iteration

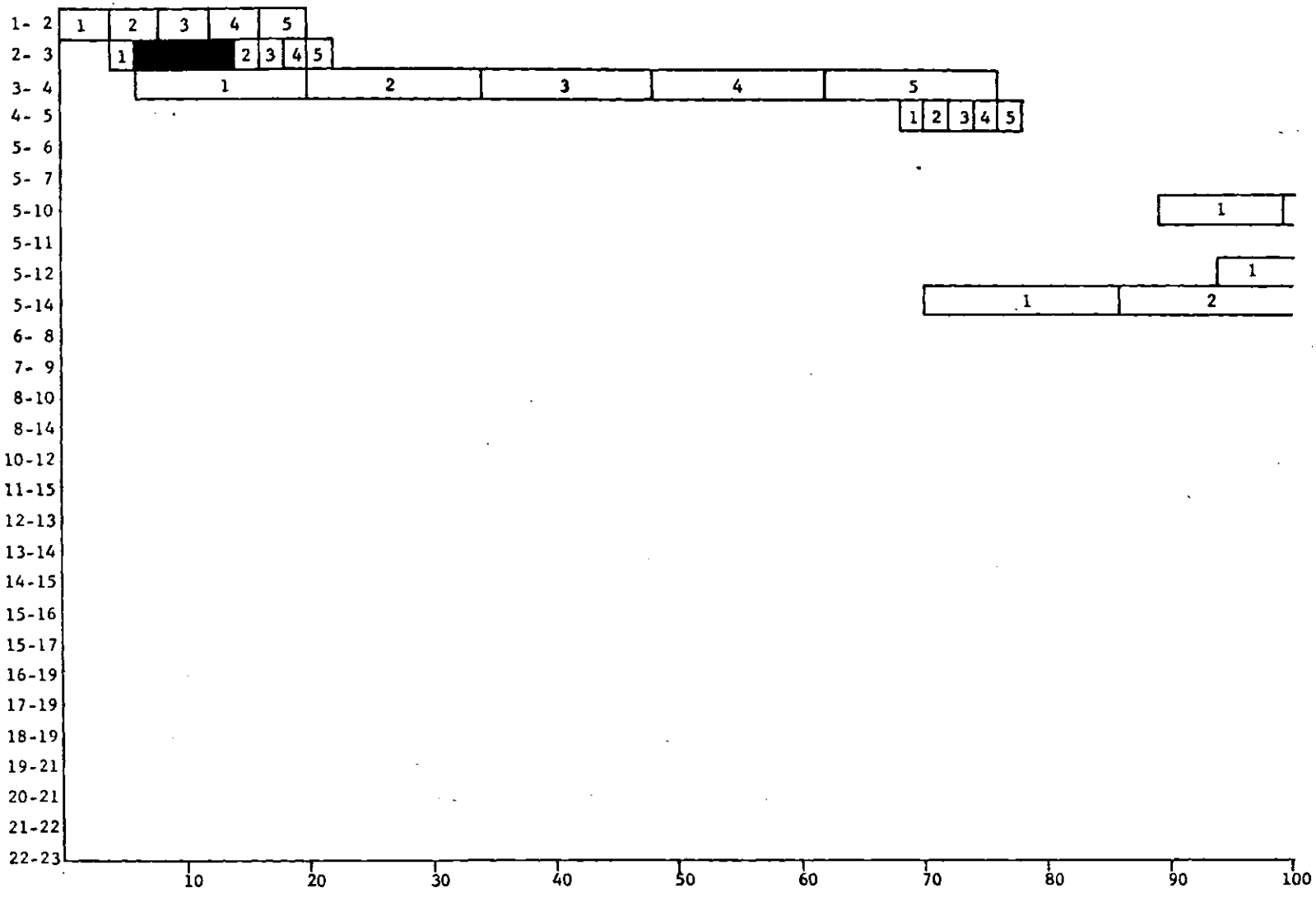


Figure 31. Gantt Chart After the Fifth Iteration

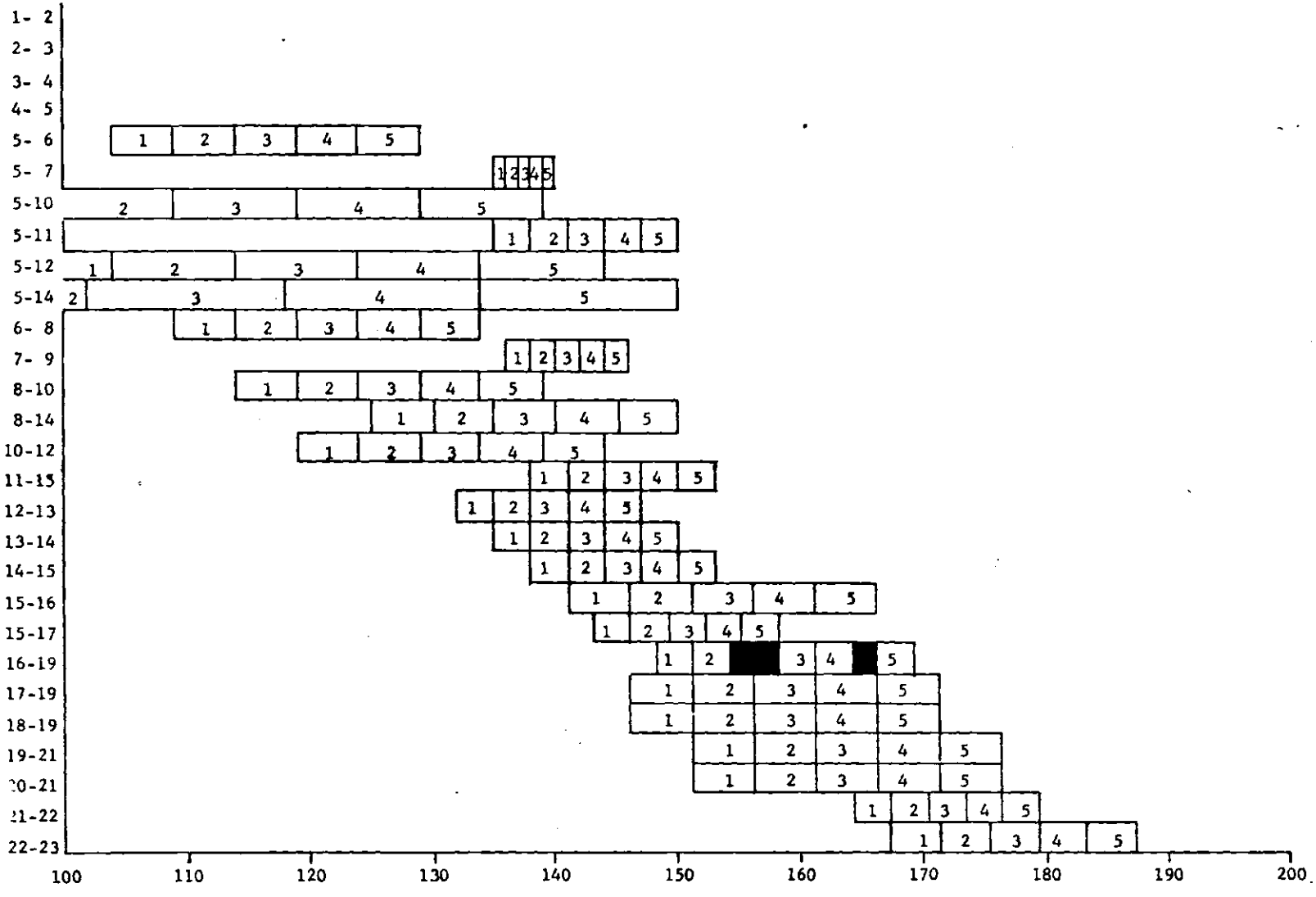


Figure 31. (Continued)

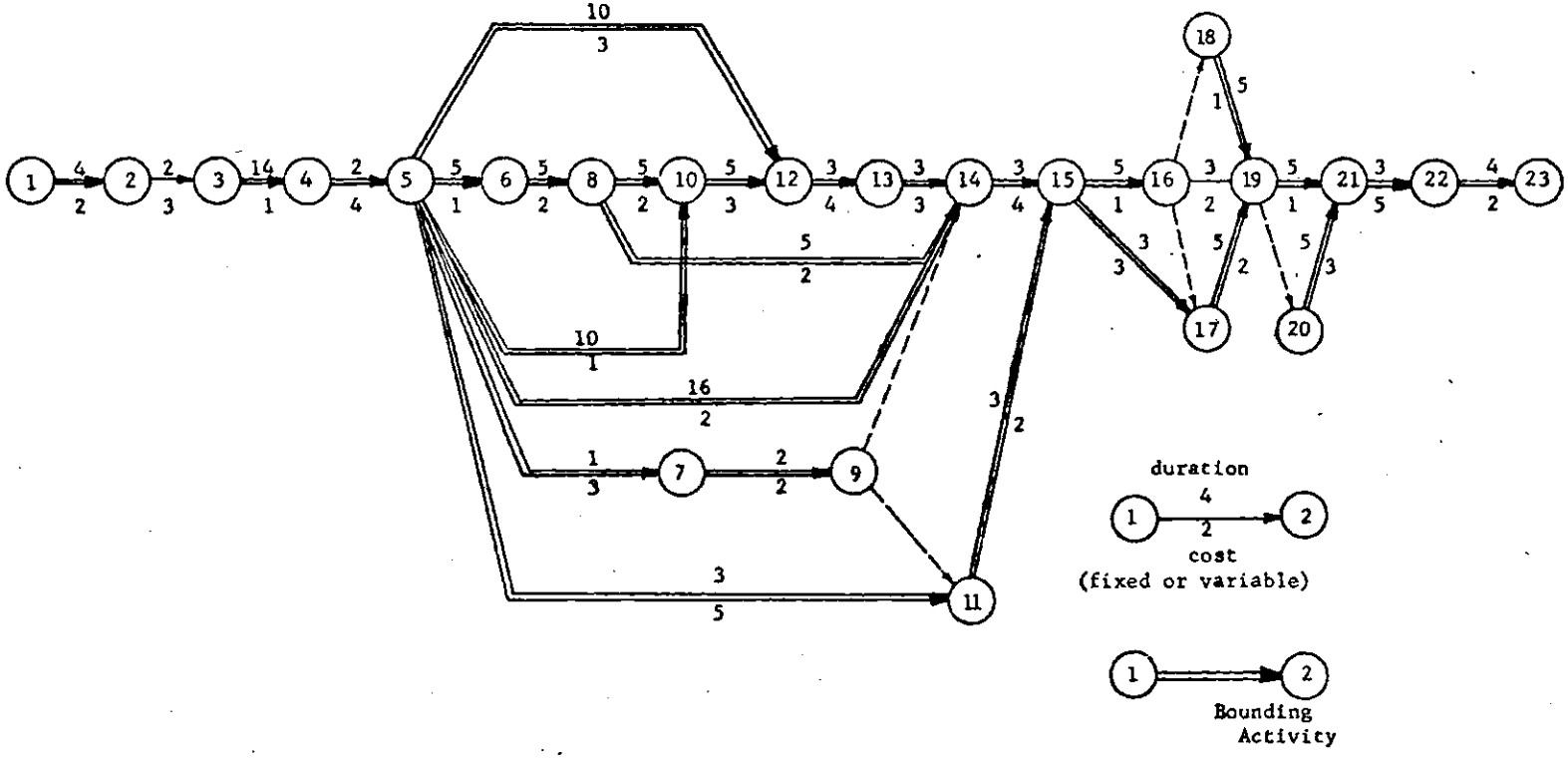


Figure 32. Bounding Activities for the Sixth Iteration

Table 10. Activity Schedules and Costs at the End of the Sixth Iteration

Activity	Schedule					Interruptions		Cost/Unit (V_i)	Total Cost
	1	2	3	4	5	Number	Length		
1 - 2	0	4	8	12	16	0	0	2	0
2 - 3	12	14	16	18	20	0	0	3	0
3 - 4	14	28	42	56	70	0	0	1	0
4 - 5	76	78	80	82	84	0	0	4	0
5 - 6	112	117	122	127	132	0	0	1	0
5 - 7	143	144	145	146	147	0	0	3	0
5 - 10	97	107	117	127	137	0	0	1	0
5 - 11	143	146	149	152	155	0	0	5	0
5 - 12	102	112	122	132	142	0	0	3	0
5 - 14	78	94	110	126	142	0	0	2	0
6 - 8	117	122	127	132	137	0	0	2	0
7 - 9	144	146	148	150	152	0	0	2	0
8 - 10	122	127	132	137	142	0	0	2	0
8 - 14	133	138	143	148	153	0	0	2	0
10 - 12	127	132	137	142	147	0	0	3	0
11 - 15	146	149	152	155	158	0	0	2	0
12 - 13	140	143	146	149	152	0	0	4	0
13 - 14	143	146	149	152	155	0	0	3	0
14 - 15	146	149	152	155	158	0	0	4	0
15 - 16	149	154	159	164	169	0	0	1	0
15 - 17	151	154	157	160	163	0	0	3	0
16 - 19	156	159	166	169	174	2	6	2	12
17 - 19	154	159	164	169	174	0	0	2	0
18 - 19	154	159	164	169	174	0	0	1	0
19 - 21	159	164	169	174	179	0	0	1	0
20 - 21	159	164	169	174	179	0	0	3	0
21 - 22	172	175	178	181	184	0	0	5	0
22 - 23	175	179	183	187	191	0	0	2	0

Total = 12

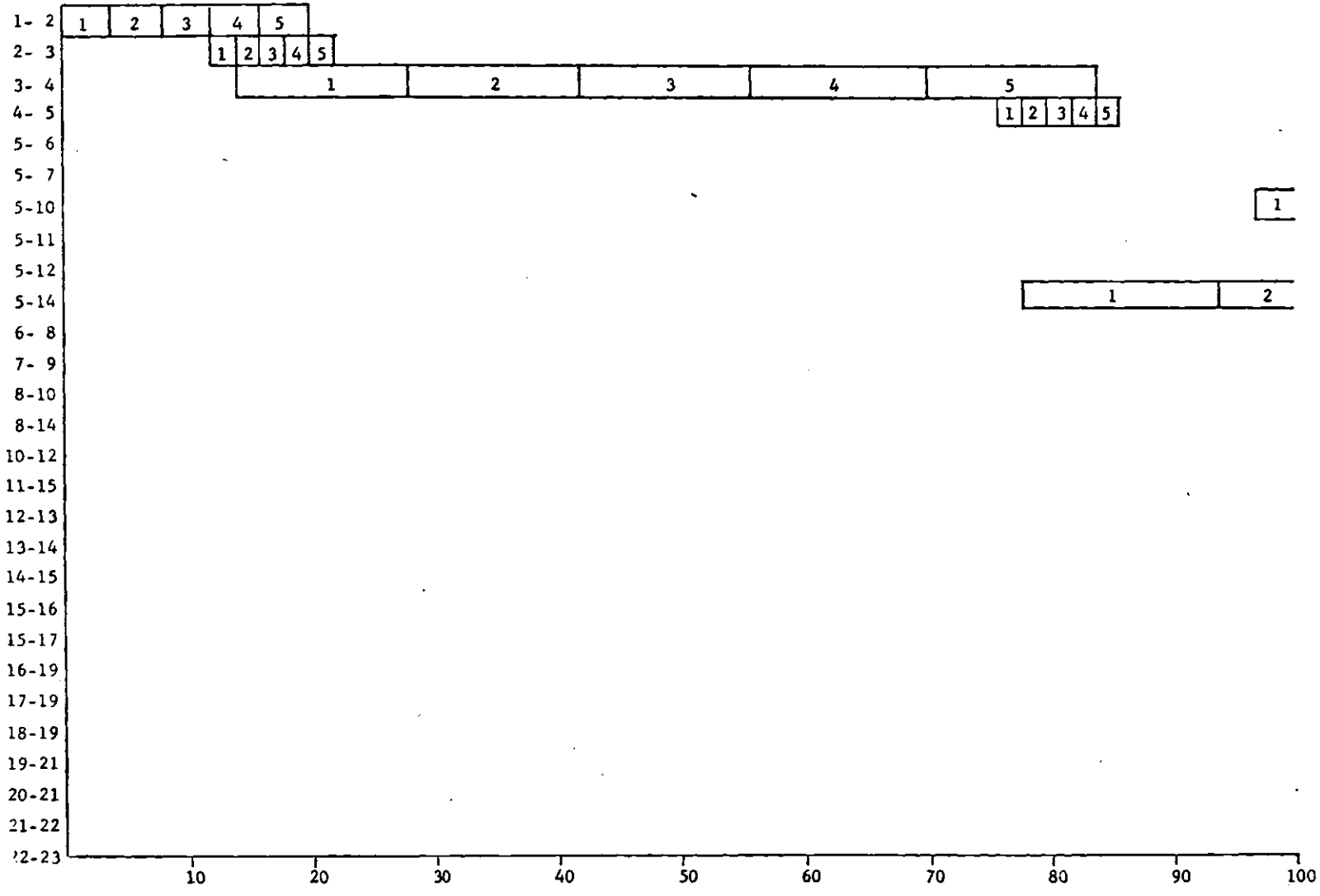


Figure 33. Gantt Chart After the Sixth Iteration

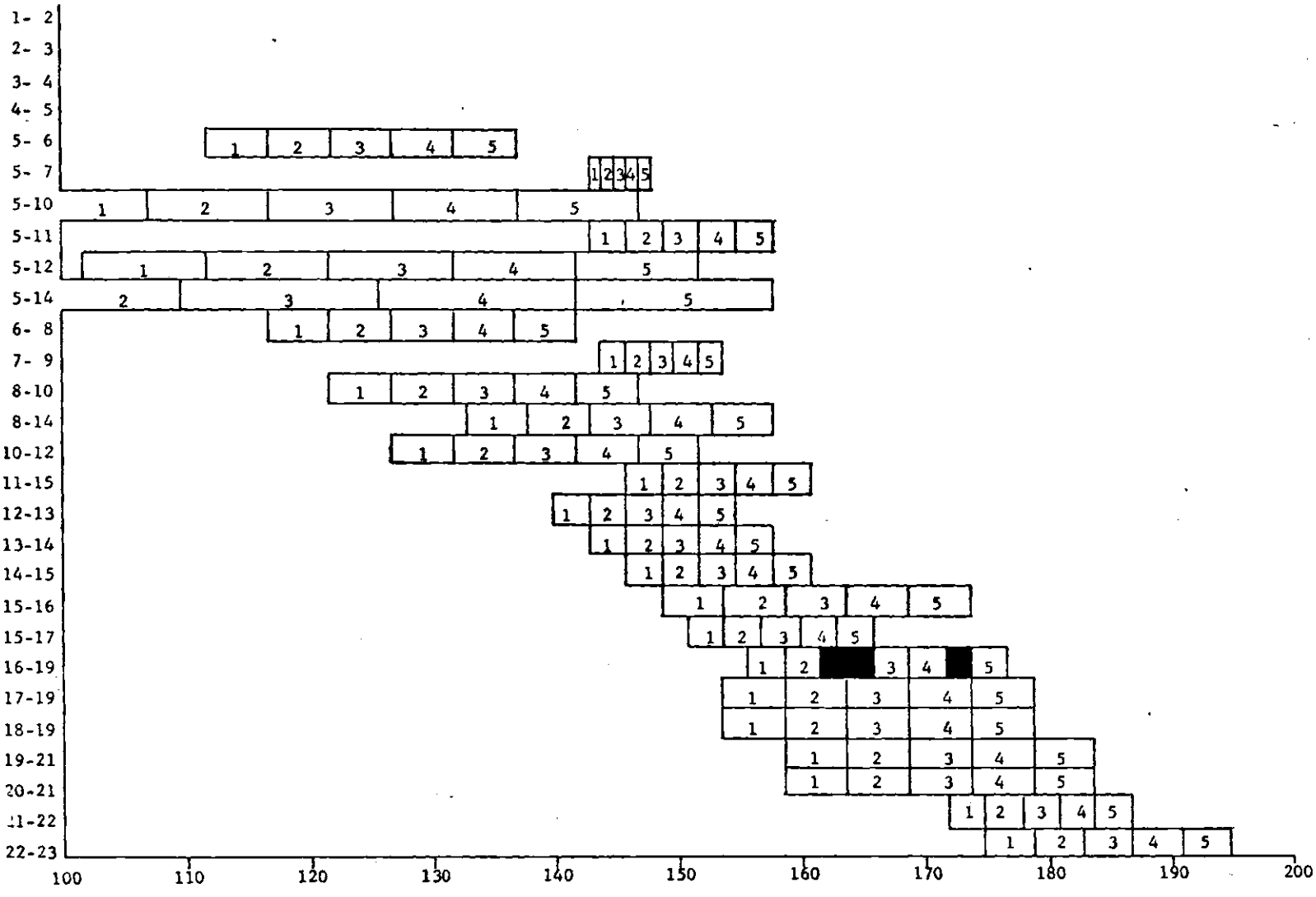


Figure 33. (Continued)

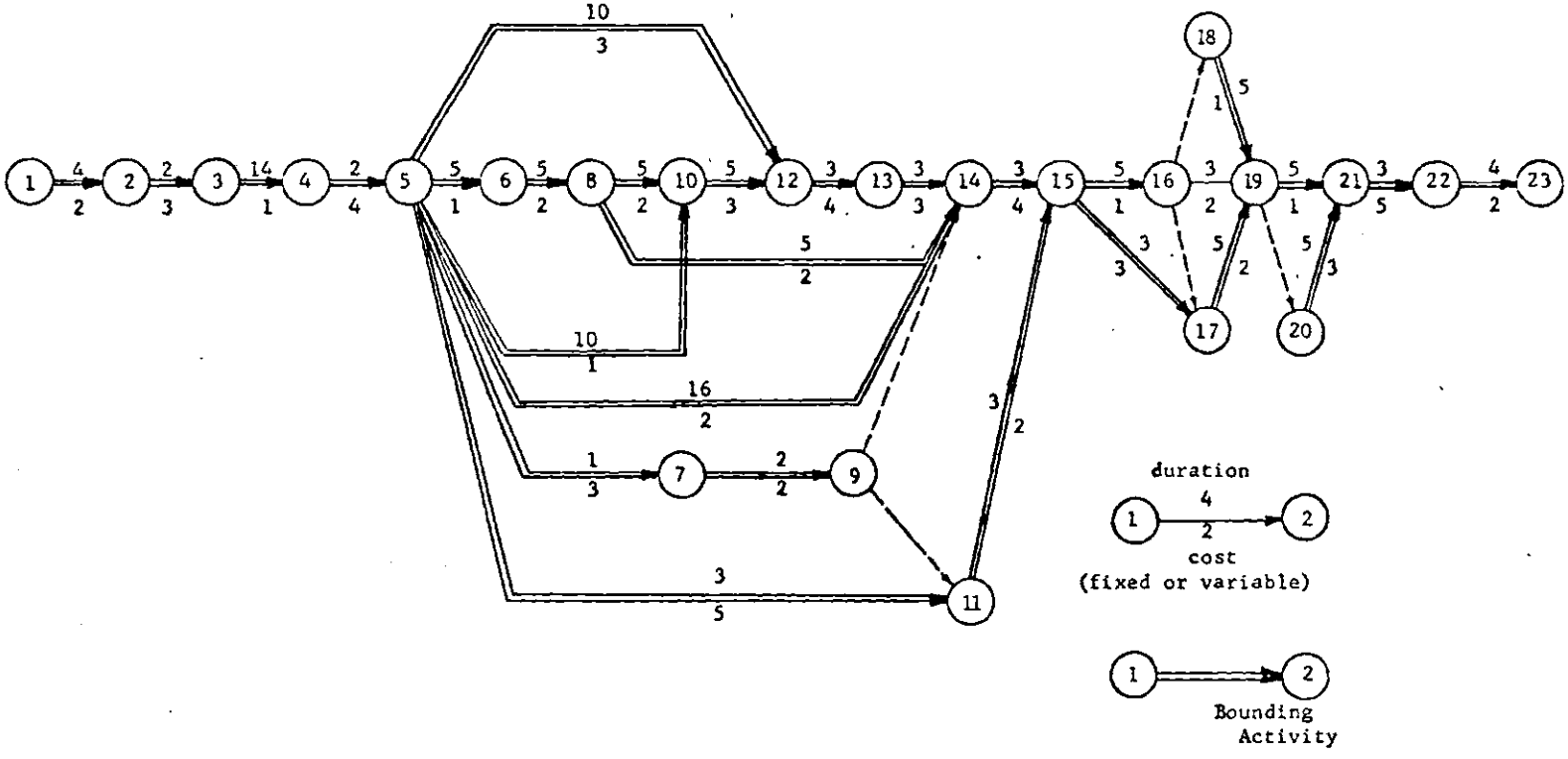


Figure 34. Bounding Activities for the Seventh Iteration

Table 11. Activity Schedules and Costs at the End of the Seventh Iteration

Activity	Schedule					Interruptions		Cost/Unit	Total
	1	2	3	4	5	Number	Length	(V_i)	Cost
1 - 2	0	4	8	12	16	0	0	2	0
2 - 3	12	14	16	18	20	0	0	3	0
3 - 4	14	28	42	56	70	0	0	1	0
4 - 5	76	78	80	82	84	0	0	4	0
5 - 6	112	117	122	127	132	0	0	1	0
5 - 7	143	144	145	146	147	0	0	3	0
5 - 10	97	107	117	127	137	0	0	1	0
5 - 11	143	146	149	152	155	0	0	5	0
5 - 12	102	112	122	132	142	0	0	3	0
5 - 14	78	94	110	126	142	0	0	2	0
6 - 8	117	122	127	132	137	0	0	2	0
7 - 9	144	146	148	150	152	0	0	2	0
8 - 10	122	127	132	137	142	0	0	2	0
8 - 14	133	138	143	148	153	0	0	2	0
10 - 12	127	132	137	142	147	0	0	3	0
11 - 15	146	149	152	155	158	0	0	2	0
12 - 13	140	143	146	149	152	0	0	4	0
13 - 14	143	146	149	152	155	0	0	3	0
14 - 15	146	149	152	155	158	0	0	4	0
15 - 16	149	154	159	164	169	0	0	1	0
15 - 17	151	154	157	160	163	0	0	3	0
16 - 19	162	165	168	171	174	0	0	2	0
17 - 19	160	165	170	175	180	0	0	2	0
18 - 19	160	165	170	175	180	0	0	1	0
19 - 21	165	170	175	180	185	0	0	1	0
20 - 21	165	170	175	180	185	0	0	3	0
21 - 22	178	181	184	187	190	0	0	5	0
22 - 23	181	185	189	193	197	0	0	2	0

Total = 0

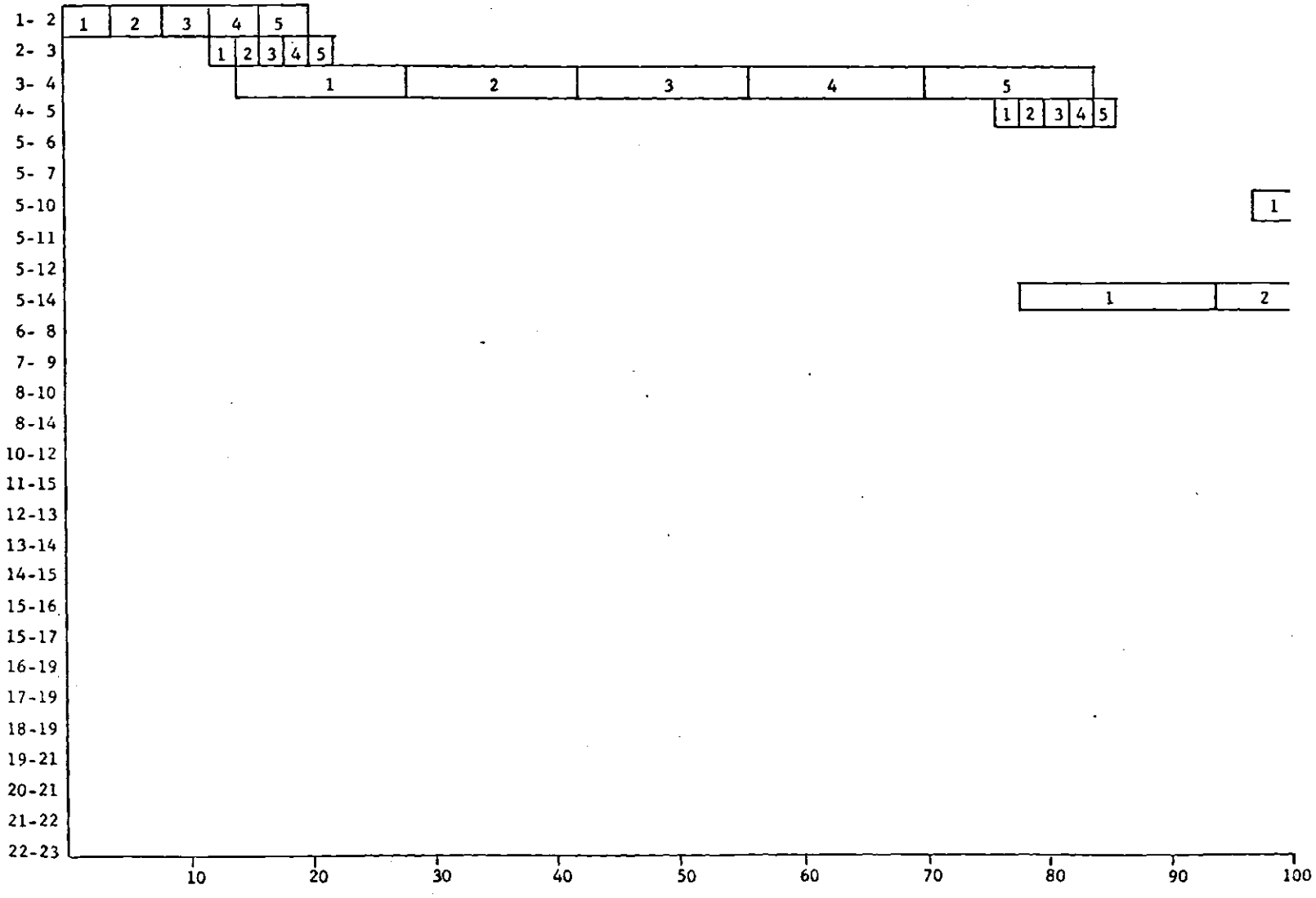


Figure 35. Gantt Chart After the Seventh Iteration

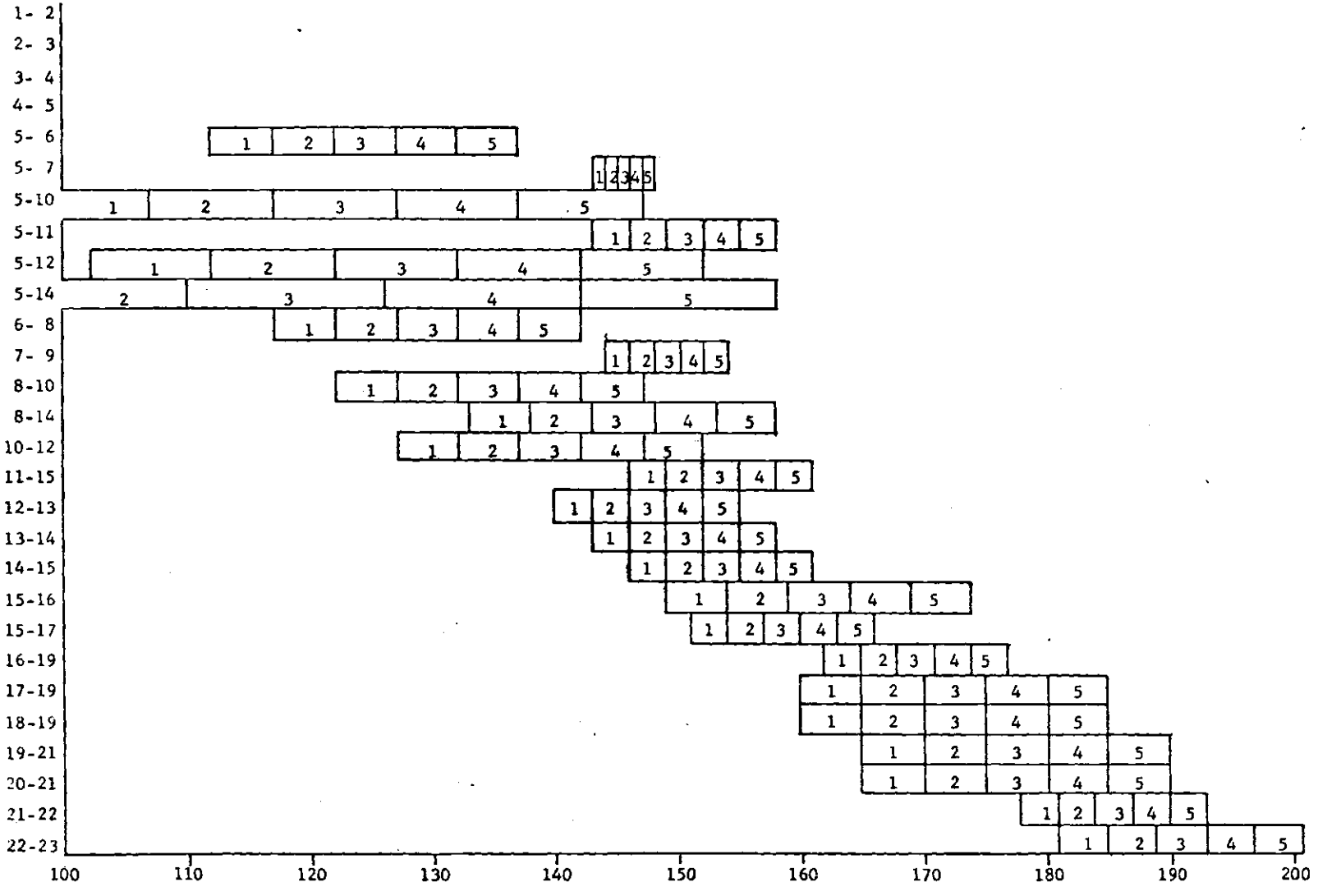


Figure 35. (Continued)

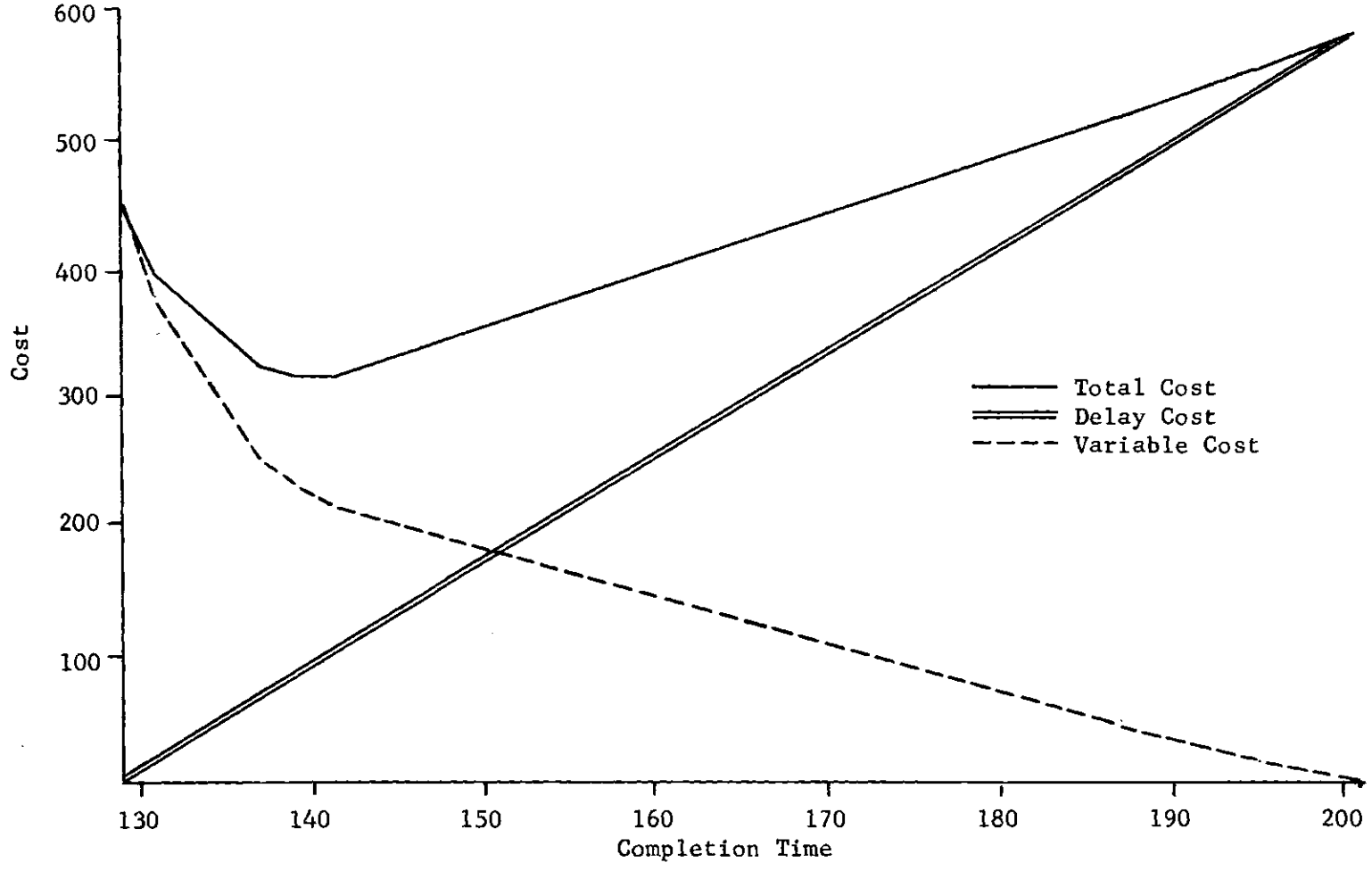


Figure 36. Cost Curve Resulting from the Application of the Time-Cost Trade-Off Procedure to the Schedule of Figure 8

APPENDIX B

DOCUMENTATION OF THE BATCH MODE SCHEDULING PROGRAM

This appendix gives the documentation for the batch mode cyclic project scheduling program CPSS, which was described in Chapter VI. This documentation consists of a listing of the important variables and arrays, a summary of the subroutines, a description of input and output formats, a program listing, and a sample run using the network of Figure 3 as an example.

Important Variables and Arrays

Some of the important variables and arrays used by the scheduling program are listed below. Other variables which occur in the program listing represent variables which are local to a particular subroutine and which are used mainly as indices, indicators, and temporary storage areas. Their usage may be determined from the context of the program statements. The major variables, all of type INTEGER, are:

CHART - indicator variable used to select the option of calling CYCCHT, which prints a Gantt chart of the computed schedule. If this option is desired, CHART is set equal to the value 1.

COUNT - a five-element array which maintains a count of the number of unscheduled activities remaining in the subsets L, L', M, M', and K, respectively.

D - a one-dimensional array of 100 elements which stores the duration of all activities in the project.

DCOST - stores the constant representing the per unit project delay cost.

DESC - a two-dimensional array which stores, for each activity, an alpha-numeric description of the activity which is up to twenty-four characters in length.

ESS - a two-dimensional array which stores the early start times for each cycle of every activity in the project.

FLAG - a one-dimensional array of 100 elements which indicates the scheduling status of each activity. This is indicated as follows:

FLAG(I) = 0: activity I is scheduled;

FLAG(I) = 1: activity I is in set R;

FLAG(I) = 2: activity I is in set L;

FLAG(I) = 3: activity I is in set L';

FLAG(I) = 4: activity I is in set M;

FLAG(I) = 5: activity I is in set M';

FLAG(I) = 6: activity I is in set K.

I - one-dimensional, 100 element array which stores the i-nodes of all activities in the network.

ICOSTF - one-dimensional, 100 element array which stores the fixed cost of interruption for each activity.

ICOSTV - one-dimensional, 100 element array which stores the variable cost of interruption for each activity.

IND1 - an indicator variable. If set to 1, the subroutine TRACE will be called and a trace of the scheduling steps will be printed.

IND2 - an indicator variable. If set to 1, the subroutine TIMWRT will write the early and late start schedules for the cyclic project.

IND3 - an indicator variable. If set to 1 when IND2 is also set to 1, TIMWRT and CYCCHT will print a Gantt chart of the early and late start schedules for the project.

J - a one-dimensional, 100 element array which stores the j-nodes for each activity in the network.

LENGTH - the computed length of interruptions in a schedule of an activity. This value is computed and returned by the subroutine LOWBND.

LIST - indicator variable. If set to 1, the subroutine CYCLST will be called and a listing of the project schedule and costs will be printed.

LOW - the computed number of interruptions in the schedule of an activity. This value is computed and returned by the subroutine LOWBND.

LSS - a two-dimensional array which stores the late start time for each cycle of every activity in the project.

MAXD - if specified on input to the program, this will be the completion time for the project. Otherwise, this will be the computed minimum completion time for the project.

NACTS - the number of activities in the project, including dummy activities.

NCYC - the number of cycles in the project.

ORDER - the set indicating the order of the activities based upon the sum of their i-node and j-node values.

PCTN - fifty element array which stores the number of unscheduled activities entering each node.

RANDOM - an indicator variable which is set to 1 if a random network is to be generated.

S - a two-dimensional array which stores the computed schedule for each cycle of every activity.

SCHCNT - contains, at any time, the total number of activities in the project which have been scheduled.

SCNT - fifty element array which stores the number of unscheduled activities leaving each node.

SEED - stores the initial value of the seed for the random number generator when a random network is computed. Used only when RANDOM is set equal to 1.

SPAN - indicator variable. If SPAN equals 1, the span-reducing subroutine SHIFT is called when using the fixed-cost objective function.

STAR - a one-dimensional, 100 element array which indicates which activities must be continuous in the schedule. If STAR(I) is set equal to an asterisk ('*') on input, the activity will be scheduled continuously.

STRCNT - stores the number of activities in the project which must be continuous.

TESS - temporary storage area for early start time values.

TLSS - temporary storage area for late start time values.

TYPE - indicator variable used to select the type of objective function to be used in computing the schedule. This is indicated as:

TYPE = 0 for fixed cost only;

TYPE = 1 for variable cost only;

TYPE = 2 for sum of fixed and variable cost;

TYPE = 3 for number of interruptions;

TYPE = 4 for sum of squares of interruption lengths.

Program Subroutines

CYCLIC - the main scheduling subroutine in the program. Computes the low-cost schedule for the project and returns start times of the activities to the main program in the array S.

UPSET - called by subroutine CYCLIC. This subroutine, called after the scheduling of any activity by CYCLIC, updates the arrays COUNT and FLAG, which indicate the set membership and scheduling status of all activities in the network.

UPCNT - subroutine called by UPSET. This subroutine updates the arrays PCNT and SCNT as activities are scheduled so that UPSET may properly determine the status of activities.

CHGCNT - changes the values of elements of PCNT and SCNT to reflect the precedence relationships of dummy activities.

CYCESS - computes the early start schedule for any activity.

CYCLSS - computes the late start schedule for any activity.

BACALG - computes the backward algorithm schedule for any activity.

FORALG - computes the forward algorithm schedule for any activity.

ICLM - a major subroutine called by CYCLIC. Performs the computation of the lower bounds ICL and ICM and schedules activities out of the sets L' and M'.

SHIFT - performs the span-reducing shifting operations.

INIT1 - a subroutine called only by SHIFT. This subroutine initiates

the S array elements for dummy activities with the minimum start times of their successor activities.

INIT2 - another subroutine called only by SHIFT. This subroutine initiates the S array elements for dummy activities with the maximum completion times for their predecessor activities.

RANGEN - if random network generation is specified, this subroutine is called by the main program before entering CYCLIC in order to generate a random network. All the generated values are entered into the proper arrays and variables and are returned to the main program.

GEN - random number generator used by RANGEN.

TRACIT - a subroutine which may optionally be called by CYCLIC or ICLM after the scheduling of every activity. TRACIT will provide a trace of the scheduling procedure, showing the status of each of the unscheduled subsets, the subset from which the currently scheduled activity is being removed, the activity's schedule, and the unscheduled predecessor and successor activity counts for each node.

TIMWRT - another subroutine which may be optionally called by CYCLIC. This subroutine will print a listing of the ESS and LSS schedules as well as provide Gantt charts of each.

CHECK - a subroutine called by the main program which determines if the computed schedule is totally feasible. If the schedule is not feasible, appropriate error messages are printed.

CYCLST - writes a formatted listing of the activity schedules and resulting costs. May be optionally called by the main program.

CYCCHT - writes a formatted Gantt chart of the activity schedules. May be optionally called by the main program.

Input Formats

Input to the program is provided via standard eighty column punched cards. The first card is a control card which is followed by a series of cards, one for each activity in the project.

Card 1

The first card is the control card. The information provided on this card contains all the necessary information for controlling the execution of the program. The format used is the FORTRAN V NAMELIST format. The first column of the card is left blank. The characters $INPUT are punched in columns two through seven. A blank is then left in column eight and the variable names are listed along with their values. Each variable name is punched, along with an equals sign (=) and the value for the variable. Variables which are to have a zero value may be omitted from the NAMELIST. Acceptable variables and the limits on their values are:

CHART: 0 or 1;

DCOST: any integer value;

IND1: 0 or 1;

IND2: 0 or 1;

IND3: 0 or 1;

LIST: 0 or 1;

MAXD: 0 or any completion time that is greater than the minimum completion time and less than or equal to the maximum

Page missing from thesis

columns 4-5: enter the j-node for the activity, a number between 1 and 50;

columns 6-29: enter, if desired, an alphanumeric description of the activity;

columns 30-32: enter an integer for the fixed interruption cost for the activity.

columns 33-35: enter an integer for the variable interruption cost for an activity;

columns 36-80: blank.

Output Formats

Two major outputs may be optionally printed. These are a listing of the activity schedules and resulting costs and a Gantt chart of the project schedule.

Activity Schedule Listing

This output lists sequentially the activities in the project. For each activity, the start and finish time of each cycle is listed. When all activities have been listed, a summary of project costs is printed. This summary lists the activities sequentially and indicates the frequency and length of interruptions in the schedule. The resulting interruption and delay costs are also calculated. A summary of the total costs of interruption, delay, and total project costs is also given. The format of the output is such that it is completely self explanatory.

Gantt Chart Output

If desired, a Gantt chart of the schedule may also be printed. Several of these are used in Chapters I, III, and IV. The time scale

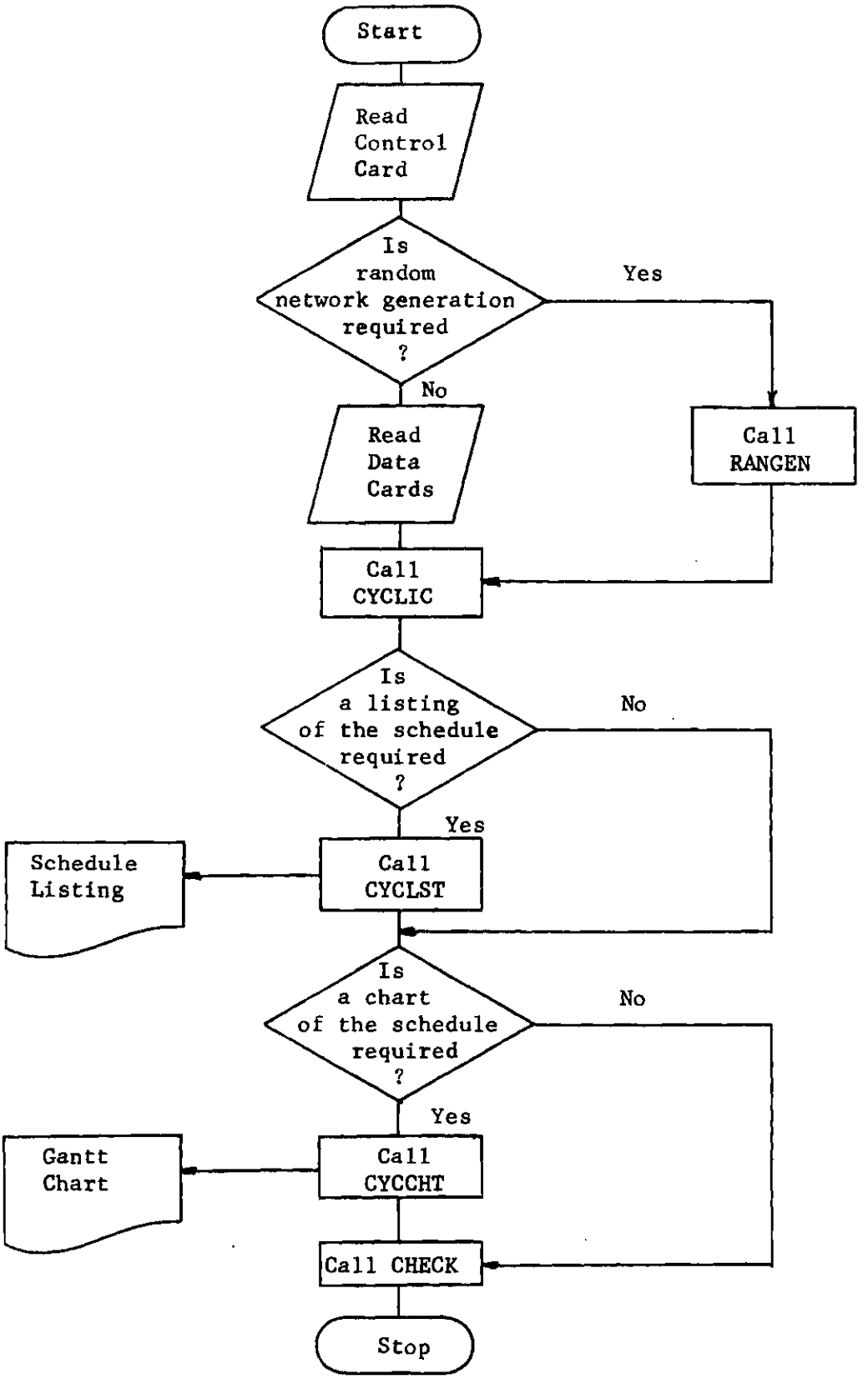
is along the horizontal axis with one print space representing a unit of time. The scale is indicated along the top of the chart.

The schedule of each activity is printed on one line. The i-node and j-node of each activity are listed along with the first cycle of the project which is printed on that line. This value, listed under the columns labeled "START CYCLE," indicates the cycle that is being represented by the first series of numbers in the chart. For example, if 23 is the value of the starting cycle for a particular activity and the first series of numbers is a series of 3's, this indicates that the 3's form a bar which represents the schedule of the twenty-third cycle of the activity.

It may be necessary to print multi-page outputs in order to form the Gantt chart for the entire project schedule. The charts are printed in blocks of 100 time units. For the first hundred time units, the schedules of all activities are printed, using multiple pages if the number of activities in the project is large. Then, the next hundred time units are considered, etc. The program has a capacity to print a schedule for fifty cycles of a 100 activity project with a completion time less than or equal to 9999.

Flowchart, Program Listing, and Sample Output

The following pages represent, in order, a macro flowchart of the execution of the CPSS program, a complete source listing of the program, and sample input and output for the problem of Figure 3.


Start
Read Control Card
Is random network generation required ?
Yes
No
Call RANGEN
Read Data Cards
Call CYCLIC
Is a listing of the schedule required ?
No
Yes
Schedule Listing
Call CYCLST
Is a chart of the schedule required ?
No
Yes
Gantt Chart
Call CYCCHT
Call CHECK
Stop

```
      IMPLICIT INTEGER(A-Z)
C***VARIABLES
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      COMMON /FOUR/STAR(100),STRCNT
      COMMON /FIVE/ESS(100,50),LSS(100,50)
      COMMON /SIX/PCNT(50),SCNT(50),FLAG(100),COUNT(5)
      COMMON /SEVEN/TCOSTF(100),ICOSTV(100),DESC(100,4),
     *DCOST
      COMMON /EIGHT/PAGE
      COMMON /NINE/SSET(100)
      DIMENSION S(100,50)
      DATA DESC/400*'......'/
      NAMELIST /INPUT/NACTS,NCYC,RANDOM,IND1,IND2,IND3,IND4,
     *IND5,SPAN,SEED,DCOST,CHART,LIST,MAXD,TYPE
C***READ CONTROL CARD
      READ(5,INPUT)
      IF(RANDOM-1) 10,150,150
C***READ CARD INPUT
   10 IF(NACTS-100) 30,30,20
   20 WRITE(6,5000)
5000  FORMAT(5X,'ERROR-NUMBER OF ACTIVITIES GT 100')
      GO TO 9999
   30 IF(NCYC-50) 50,50,40
   40 WRITE(6,5001)
5001  FORMAT(5X,'ERROR-NUMBER OF CYCLES GT 50')
      GO TO 9999
   50 MAXD=1
      DO 140 N=1,NACTS
         READ(5,1000)STAR(N),I(N),J(N),(DESC(N,K),K=1,4),
     *   D(N),TCOSTF(N),ICOSTV(N)
1000     FORMAT(A1,2(I2),4A6,3(I3))
         IF(STAR(N).EQ.'*') STRCNT=STRCNT+1
         IF(I(N)-J(N)) 70,60,70
   60    WRITE(6,5002)I(N),J(N)
5002     FORMAT(5X,'ERROR-NODE I= ',I2,' GE NODE J= ',I2)
         GO TO 9999
   70    IF(N-1) 130,130,80
   80    IF(I(N)-I(N-1)) 90,100,130
   90    WRITE(6,5003)I(N-1),J(N-1),I(N),J(N)
5003     FORMAT(5X,'ERROR-I NODES NOT ARRANGED IN ASCEND',
     *        'ING ORDER',/,5X,'I= ',I2,' J= ',I2,' I= ',
     *         I2,' J= ',I2)
         GO TO 9999
  100    IF(J(N)-J(N-1)) 110,120,130
  110    WRITE(6,5004)I(N)
5004     FORMAT(5X,'ERROR-J NODES WITHIN NODE I= ',I2,
     *        ' NOT ARRANGED IN ASCENDING ORDER')
         GO TO 9999
  120    WRITE(6,5005)I(N),J(N)
```

```
 5005       FORMAT(5X,'ERROR-2 ACTIVITIES FROM NODE I= ',I2,
     *            ' TO NODE J= ',I2)
            GO TO 9999
  130       II=I(N)
            JJ=J(N)
C***FORM THE NODE COUNTS
            SCNT(II)=SCNT(II)+1
            PCNT(JJ)=PCNT(JJ)+1
  140   CONTINUE
C***CARRY OUT THE ALGORITHM
  150   IF(RANDOM.EQ.1) CALL RANGEN(SEED,NSEED,IND4,IND5)
        IF(RANDOM.EQ.1) WRITE(6,5006)SEED
 5006   FORMAT(1X,'THE SEED FOR THIS PROBLEM IS ',I6)
        CALL CYCLIC(S,MAXD,TYPE,IND1,IND2,IND3,SPAN)
        IF(LIST.EQ.1) CALL CYCLST(S,ESS(NACTS,NCYC))
        IF(CHART.EQ.1) CALL CYCCHT(S,MAXD)
        CALL CHECK(S)
        IF(RANDOM.EQ.1) WRITE(6,5007)NSEED
 5007   FORMAT(1X,'THE SEED FOR THE NEXT PROBLEM IS ',I6)
 9999   STOP
        END
C***********************************************************
        SUBROUTINE CYCLIC(S,MAXD,TYPE,IND1,IND2,IND3,SPAN)
        IMPLICIT INTEGER(A-Z)
C***
C***VARIABLES
        COMMON /ONE/I(100),J(100)
        COMMON /TWO/NACTS,NCYC
        COMMON /THREE/D(100)
        COMMON /FOUR/STAR(100),STRCNT
        COMMON /FIVE/ESS(100,50),LSS(100,50)
        COMMON /SIX/PCNT(50),SCNT(50),FLAG(100),COUNT(5)
        COMMON /SEVEN/ICOSTF(100),ICOSTV(100),DESC(100,4),
       *DCOST
        COMMON /NINE/SSET(100)
        DIMENSION S(100,50),TESS(2,50),TLSS(2,50),ORDER(100)
C***
        SCHCNT=0
        MAXD2=0
        DU=0
        DO 5 A=1,100
             FLAG(A)=1
             SSET(A)=0
             DO 4 B=1,NCYC
                  ESS(A,B)=0
                  LSS(A,B)=0
                  S(A,B)=0
   4         CONTINUE
   5    CONTINUE
C***ARRANGE ACTIVITIES BY INCREASING I+J
        SIZE=I(NACTS)+J(NACTS)
```

```
      N=1
      DO 20 A=3,SIZE
           DO 10 B=1,NACTS
                IF(D(B).EQ.0) DUM=1
                IF((I(B)+J(B)).NE.A) GO TO 10
                ORDER(N)=B
                N=N+1
 10         CONTINUE
 20   CONTINUE
C***
      IF(DUM.EQ.1) CALL CHGCNT
C***FIND ESS FOR ALL ACTIVITIES
      DO 30 N=1,NACTS
           B=ORDER(N)
           CALL CYCESS(B)
           IF(J(B).EQ.J(NACTS))
     *          MAXD2=MAX(MAXD2,(ESS(B,NCYC)+D(B)))
 30   CONTINUE
C***
C***SCHEDULE THOSE ACTIVITIES WHICH MUST BE CONTINUOUS
      IF(STRCNT) 40,120,40
 40   DO 70 N=1,NACTS
           B=ORDER(N)
           IF(STAR(B).NE.'*') GO TO 70
           S(B,NCYC)=ESS(B,NCYC)
           LSS(B,NCYC)=S(B,NCYC)
           M=NCYC-1
           DO 50 CYC=M,1,-1
                S(B,CYC)=S(B,CYC+1)-D(B)
                ESS(B,CYC)=S(B,CYC)
                LSS(B,CYC)=S(B,CYC)
 50        CONTINUE
           NN=N+1
           DO 60 M=NN,NACTS
                C=ORDER(M)
                CALL CYCESS(C)
                IF(J(C).EQ.J(NACTS))
     *               MAXD=MAX(MAXD,(ESS(C,NCYC)+D(C)))
 60        CONTINUE
           FLAG(B)=0
           CALL UPCNT(B,DUM)
           SCHCNT=SCHCNT+1
           IF(SCHCNT.EQ.STRCNT) GO TO 120
 70   CONTINUE
C***
C***TRY TO ELIMINATE REMAINING INTERRRUPTIONS
 90   DO 110 N=1,NACTS
           B=ORDER(N)
           IF(STAR(B).EQ.'*') GO TO 110
           DO 100 CYC=2,NCYC
                IF((ESS(B,CYC)-ESS(B,CYC-1)).GT.D(B))
```

```
     *                GO TO 120
  100       CONTINUE
  110  CONTINUE
       GO TO 510
C***
C***FIND LSS TIMES FOR ACTIVITIES
  120  IF(MAXD.EQ.0) MAXD=MAXD2
       DO 150 N=NACTS,1,-1
            B=ORDER(N)
            IF(STAR(B).EQ.'*') GO TO 150
            CALL CYCLSS(B,MAXD)
            IF(D(B).EQ.0) GO TO 140
            DO 130 CYC=1,NCYC
                 IF(ESS(B,CYC).NE.LSS(B,CYC)) GO TO 150
  130       CONTINUE
            DO 135 CYC=1,NCYC
                 S(B,CYC)=ESS(B,CYC)
  135       CONTINUE
            FLAG(B)=0
            CALL UPCNT(B,DUM)
            SCHCNT=SCHCNT+1
            GO TO 150
  140       SCHCNT=SCHCNT+1
  150  CONTINUE
       IF(IND2.EQ.1) CALL TIMWRT(IND3,MAXD)
C***
C***
C***FORM THE SETS OF UNSCHEDULED ACTIVITIES
       CALL SET
C***
C***TRY TO SCHEDULE ALL ACTIVITIES IN THE SET L
  160  IF(COUNT(1)) 170,310,170
  170  DO 300 N=1,NACTS
            B=ORDER(N)
            IF(FLAG(B)-2) 300,180,300
  180       ASTER=0
C***FIND THE ES FOR THE ACTIVITY
            CALL PACALG(B,S)
C***SAVE THE CURRENT ESS OF B IN TEMPORARY STORAGE
C*** AND SET ESS OF B TO ITS COMPUTED ES
            DO 190 CYC=1,NCYC
                 TESS(1,CYC)=ESS(B,CYC)
                 ESS(B,CYC)=S(B,CYC)
  190       CONTINUE
C***SEARCH FOR EVERY SUCCESSOR OF B
            DO 260 M=2,NACTS
                 IF(J(B)-I(M)) 260,200,260
C***SAVE THE CURRENT ESS OF THE
C*** SUCCESSOR IN TEMPORARY STORAGE
  200            DO 210 CYC=1,NCYC
                      TESS(2,CYC)=ESS(M,CYC)
```

```
 210              CONTINUE
C***COMPUTE THE ESS OF THE SUCCESSOR M
C*** WITH ESS OF B EQUAL TO BS
                  CALL CYCESS(M)
C***CHECK TO SEE IF ESS TIMES HAVE  CHANGED
                  DO 230 CYC=1,NCYC
                       IF(ESS(M,CYC)-TESS(2,CYC)) 220,230,220
 220                   ASTER=1
                       GO TO 240
 230              CONTINUE
                  GO TO 260
C***RESTORE ESS TIMES OF M, IF CHANGED
 240              DO 250 CYC=1,NCYC
                       ESS(M,CYC)=TESS(2,CYC)
 250              CONTINUE
 260         CONTINUE
C***RESTORE ESS OF B
             DO 270 CYC=1,NCYC
                  ESS(B,CYC)=TESS(1,CYC)
 270         CONTINUE
             IF(ASTER) 280,290,280
C***IF ESS OF ANY SUCCESSOR CHANGED,
C*** PUT B IN SET L'
 280         FLAG(B)=3
             COUNT(2)=COUNT(2)+1
             COUNT(1)=COUNT(1)-1
             IF(IND1.EQ.1) CALL TRACIT(280,'L','L'',B,S)
             GO TO 300
C***IF ESS OF ALL SUCCESSORS UNCHANGED, SCHEDULE B IN
C*** BS, REMOVE FROM L, UPDATE THE SETS
 290         COUNT(1)=COUNT(1)-1
             SCHCNT=SCHCNT+1
             SSET(B)=1
             CALL UPSET(B,DUM)
             IF(IND1.EQ.1) CALL TRACIT(290,'L','S',B,S)
             GO TO 160
 300   CONTINUE
C***
C***TRY TO SCHEDULE ALL ACTIVITIES IN THE SET L'
 310   IF(COUNT(3)) 320,460,320
 320   DO 450 N=NACTS,1,-1
             B=ORDER(N)
             IF(FLAG(B)-4) 450,330,450
 330         ASTER=0
C***FIND THE FS FOR THE ACTIVITY
             CALL FORALG(B,S)
C***SAVE THE CURRENT LSS OF B IN TEMPORARY STORAGE
C*** AND SET LSS OF B TO ITS COMPUTED BS
             DO 340 CYC=1,NCYC
                  TLSS(1,CYC)=LSS(B,CYC)
                  LSS(B,CYC)=S(B,CYC)
```

```
 340        CONTINUE
C***SEARCH FOR EVERY PREDECESSOR OF B
            DO 410 M=1,B
                 IF(I(B)-J(M)) 410,350,410
C***SAVE THE CURRENT LSS OF THE PREDECESSOR
C*** IN TEMPORARY STORAGE
 350            DO 360 CYC=1,NCYC
                      TLSS(2,CYC)=LSS(M,CYC)
 360            CONTINUE
C***COMPUTE THE LSS OF THE PREDECESSOR
C*** WITH LSS OF B EQUAL TO FS
                CALL CYCLSS(M,MAXD)
C***CHECK TO SEE IF LSS TIMES HAVE CHANGED
                DO 380 CYC=1,NCYC
                      IF(LSS(M,CYC)-TLSS(2,CYC)) 370,380,370
 370                  ASTER=1
                      GO TO 390
 380            CONTINUE
                GO TO 410
C***RESTORE LSS TIMES IF THEY WERE CHANGED
 390            DO 400 CYC=1,NCYC
                     LSS(M,CYC)=TLSS(2,CYC)
 400        CONTINUE
 410        CONTINUE
C***RESTORE LSS OF B
            DO 420 CYC=1,NCYC
                 LSS(B,CYC)=TLSS(1,CYC)
 420        CONTINUE
            IF(ASTER) 430,440,430
C***IF LSS OF ANY PREDECESSOR CHANGED,
C*** PUT B IN SET M'
 430        FLAG(B)=5
            COUNT(4)=COUNT(4)+1
            COUNT(3)=COUNT(3)-1
            IF(IND1.EQ.1) CALL TRACIT(430,'M','M'',B,S)
            GO TO 450
C***IF LSS OF ALL PREDECESSORS UNCHANGED, SCHEDULE
C*** B IN FS, REMOVE FROM M, UPDATE THE SETS
 440        COUNT(3)=COUNT(3)-1
            SCHCNT=SCHCNT+1
            SSET(B)=2
            CALL UPSET(B,DUM)
            IF(IND1.EQ.1) CALL TRACIT(440,'M','S',B,S)
            GO TO 310
 450  CONTINUE
C***SCHEDULE ALL ACTIVITIES IN THE SET K
 460  IF(COUNT(5)) 470,500,470
 470  DO 490 N=1,NACTS
           B=ORDER(N)
           IF(FLAG(B)-6) 490,480,490
C***USE BACALG TO SCHEDULE ACTIVITIES IN K
```

```
 480        CALL RACALC(B,S)
            COUNT(5)=COUNT(5)-1
            SSET(B)=1
            SCHCNT=SCHCNT+1
C***UPDATE THE SETS
            CALL UPSET(B,DUM)
            IF(IND1.EQ.1) CALL TRACIT(480,'K','S',B,S)
 490  CONTINUE
C***ANY ACTIVITIES REMAINING TO BE SCHEDULED
 500  IF(SCHCNT-NACTS) 520,510,520
 510  IF(SPAN.EQ.1.OR.TYPE.EQ.1.OR.TYPE.EQ.2.OR.
     *      TYPE.EQ.3) CALL SHIFT(S,ORDER,FLAG)
      RETURN
C***SELECT A FUNCTION AND DETERMINE THE PROPER SCHEDULE
 520  CALL ICLM(S,MAXD,ORDER,$160,$310,IND1,SCHCNT,TYPE)
      END
C*************************************************************
      SUBROUTINE UPSET(B,DUM)
      IMPLICIT INTEGER(A-Z)
C***VARIABLES
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      COMMON /SIX/PCNT(50),SCNT(50),FLAG(100),COUNT(5)
C***
      FLAG(B)=0
      CALL UPCNT(B,DUM)
      ENTRY SET
      DO 120 N=1,NACTS
           IF(FLAG(N).EQ.0.OR.D(N).EQ.0) GO TO 120
  10       II=I(N)
           JJ=J(N)
           IF(PCNT(II).NE.0.OR.SCNT(JJ).NE.0) GO TO 40
           IF(FLAG(N)-1) 20,30,20
  20       K=FLAG(N)-1
           COUNT(K)=COUNT(K)-1
  30       COUNT(5)=COUNT(5)+1
           FLAG(N)=6
           GO TO 120
  40       IF(PCNT(II)) 80,50,80
  50       IF(FLAG(N).EQ.3) GO TO 120
           IF(FLAG(N)-1) 60,70,60
  60       K=FLAG(N)-1
           COUNT(K)=COUNT(K)-1
  70       FLAG(N)=2
           COUNT(1)=COUNT(1)+1
           GO TO 120
  80       IF(SCNT(JJ)) 120,90,120
  90       IF(FLAG(N).EQ.5) GO TO 120
           IF(FLAG(N)-1) 100,110,100
 100       K=FLAG(N)-1
```

```
                COUNT(K)=COUNT(K)-1
  110           FLAG(N)=4
                COUNT(3)=COUNT(3)+1
  120   CONTINUE
        RETURN
        END
C*************************************************************
      SUBROUTINE UPCNT(B,DUM)
      IMPLICIT INTEGER(A-Z)
C***VARIABLES
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      COMMON /SIX/PCNT(50),SCNT(50),FLAG(100),COUNT(5)
      DIMENSION TPC(50),TSC(50)
C***
      II=I(B)
      JJ=J(B)
      MAXJ=J(NACTS)
      DO 10 N=1,MAXJ
           TPC(N)=PCNT(N)
           TSC(N)=SCNT(N)
   10 CONTINUE
      PCNT(JJ)=PCNT(JJ)-1
      SCNT(II)=SCNT(II)-1
      IF(DUM.NE.1) GO TO 60
      DO 30 N=B,1,-1
           IF(D(N)) 30,20,30
   20      II=I(N)
           JJ=J(N)
           IF(SCNT(JJ).NE.TSC(JJ)) SCNT(II)=SCNT(II)-1
   30 CONTINUE
      DO 50 N=1,NACTS
           IF(D(N)) 50,40,50
   40      II=I(N)
           JJ=J(N)
           IF(PCNT(II).NE.TPC(II)) PCNT(JJ)=PCNT(JJ)-1
   50 CONTINUE
   60 RETURN
      END
C*************************************************************
      SUBROUTINE CHGCNT
      IMPLICIT INTEGER(A-Z)
C***VARIABLES
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      COMMON /SIX/PCNT(50),SCNT(50),FLAG(100),COUNT(5)
C***
      MAXJ=J(NACTS)
```

```
      DO 30 N=1,MAXJ
           DO 20 A=1,NACTS
                IF(D(A).NE.0) GO TO 20
                IF(J(A).NE.N) GO TO 10
                II=I(A)
                PCNT(N)=PCNT(N)+PCNT(II)-1
10              IF(I(A).NE.N) GO TO 20
                JJ=J(A)
                SCNT(N)=SCNT(N)+SCNT(JJ)-1
20         CONTINUE
30    CONTINUE
      RETURN
      END
C*************************************************************
      SUBROUTINE CYCESS(B)
      IMPLICIT INTEGER(A-Z)
C**VARIABLES
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      COMMON /FIVE/ESS(100,50),LSS(100,50)
C***
      IF(I(B)-1) 30,10,30
10    DO 20 CYC=2,NCYC
           ESS(B,CYC)=ESS(B,CYC-1)+D(B)
20    CONTINUE
      GO TO 90
30    DO 50 M=1,B
           IF(J(M)-I(B)) 50,40,50
40         ESS(B,1)=MAX(ESS(B,1),(ESS(M,1)+D(M)))
50    CONTINUE
      DO 80 CYC=2,NCYC
           ESS1=ESS(B,CYC-1)+D(B)
           DO 70 M=1,B
                IF(J(M)-I(B)) 70,60,70
60              ESS2=MAX(ESS1,(ESS(M,CYC)+D(M)))
                ESS(B,CYC)=MAX(ESS2,ESS(B,CYC))
70         CONTINUE
80    CONTINUE
90    RETURN
      END
C*************************************************************
      SUBROUTINE CYCLSS(B,MAXD)
      IMPLICIT INTEGER(A-Z)
C***VARIABLES
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      COMMON /FIVE/ESS(100,50),LSS(100,50)
C***
      IF(J(B)-J(NACTS)) 30,10,30
```

```
   10   LSS(B,NCYC)=MAXD-D(B)
        INDX=NCYC-1
        DO 20 CYC=INDX,1,-1
             LSS(B,CYC)=LSS(B,CYC+1)-D(B)
   20   CONTINUE
        GO TO 100
   30   DO 40 CYC=1,NCYC
             LSS(B,CYC)=1000000
   40   CONTINUE
        DO 60 M=NACTS,1,-1
             IF(I(M)-J(B)) 60,50,60
   50        LSS(B,NCYC)=MIN(LSS(B,NCYC),(LSS(M,NCYC)-D(B)))
   60   CONTINUE
        INDX=NCYC-1
        DO 90 CYC=INDX,1,-1
             LSS1=LSS(B,CYC+1)-D(B)
             DO 80 M=NACTS,1,-1
                  IF(I(M)-J(B)) 80,70,80
   70             LSS2=MIN(LSS1,(LSS(M,CYC)-D(B)))
                  LSS(B,CYC)=MIN(LSS2,LSS(B,CYC))
   80        CONTINUE
   90   CONTINUE
  100   RETURN
        END
C*************************************************************
        SUBROUTINE BACALG(B,S)
        IMPLICIT INTEGER(A-Z)
C**VARIABLES
        COMMON /TWO/NACTS,NCYC
        COMMON /THREE/D(100)
        COMMON /FIVE/ESS(100,50),LSS(100,50)
        DIMENSION S(100,50)
C***
        S(B,NCYC)=ESS(B,NCYC)
        INDX=NCYC-1
        DO 20 CYC=INDX,1,-1
             IF((S(B,CYC+1)-LSS(B,CYC)).LT.D(B)) GO TO 10
             S(B,CYC)=ESS(B,CYC)
             GO TO 20
   10        S(B,CYC)=S(B,CYC+1)-D(B)
   20   CONTINUE
        RETURN
        END
C*************************************************************
        SUBROUTINE FORALG(B,S)
        IMPLICIT INTEGER(A-Z)
C**VARIABLES
        COMMON /TWO/NACTS,NCYC
        COMMON /THREE/D(100)
        COMMON /FIVE/ESS(100,50),LSS(100,50)
        DIMENSION S(100,50)
```

```
C***
      S(B,1)=LSS(B,1)
      DO 20 CYC=2,NCYC
           IF((ESS(B,CYC)-S(B,CYC-1)).LT.D(B)) GO TO 10
           S(B,CYC)=LSS(B,CYC)
           GO TO 20
   10      S(B,CYC)=S(B,CYC-1)+D(B)
   20 CONTINUE
      RETURN
      END
C***********************************************************
      SUBROUTINE LOWBND(B,S,LOW,LENGTH)
      IMPLICIT INTEGER(A-Z)
C***VARIABLES
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      DIMENSION S(100,50)
C***
      LOW=0
      LENGTH=0
      INDEX=NCYC-1
      DO 10 N=1,INDEX
           DIFF=S(B,N+1)-S(B,N)-D(B)
           IF(DIFF.LE.0) GO TO 10
           LOW=LOW+1
           LENGTH=LENGTH+DIFF
   10 CONTINUE
      RETURN
      END
C***********************************************************
      SUBROUTINE ICL(S,MAXD,ORDER,D1,D2,IND1,SCHCNT,TYPE)
      IMPLICIT INTEGER(A-Z)
C***
C***VARIABLES
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      COMMON /FIVE/ESS(100,50),LSS(100,50)
      COMMON /SIX/PCNT(50),SCNT(50),FLAG(100),COUNT(5)
      COMMON /SEVEN/ICOSTF(100),ICOSTV(100),DESC(100,4),
     *DCOST
           COMMON /NINE/SSET(100)
      DIMENSION S(100,50),S1(100,50),TESS(100,50),
     *TLSS(100,50),ORDER(100),LOW1(100),LOW2(100),
     *LENGT1(100),LENGT2(100),S2(100,50)
C***
C***COMPUTING ICL
C***FIND BS FOR EACH ACTIVITY NOT IN CL
      DO 10 N=1,NACTS
           B=ORDER(N)
```

```
            IF(FLAG(B).EQ.0.OR.FLAG(P).EQ.3.OR.D(B).EQ.0)
     *           GO TO 10
            CALL BACALG(B,S)
C***CALCULATE LOWER BOUND ON INTERRUPTIONS
            CALL LOWBND(B,S,LOW1(B),LENGT1(B))
  10  CONTINUE
C***FIND BS FOR ACTIVITIES IN L'
      DO 40 N=1,NACTS
            B=ORDER(N)
            IF(FLAG(B)-3) 40,20,40
  20        CALL BACALG(B,S)
C***SET ESS OF B=BS
            DO 30 CYC=1,NCYC
                TESS(B,CYC)=ESS(B,CYC)
                 ESS(B,CYC)=S(B,CYC)
  30        CONTINUE
  40  CONTINUE
C***CALCULATE ICL
      ICL=0
      DO 60 N=1,NACTS
            B=ORDER(N)
            IF(FLAG(B).EQ.0.OR.FLAG(P).EQ.3) GO TO 60
C***SAVE ESS IN TEMPORARY STORAGE
            DO 50 CYC=1,NCYC
                TESS(B,CYC)=ESS(B,CYC)
  50        CONTINUE
C***FIND ESS OF ACTIVITIES IN R IF ACTIVITIES IN
C*** L' HAVE ESS=BS
            CALL CYCESS(B)
C***FIND NUMBER OF INTERRUPTIONS
            IF(D(B).EQ.0) GO TO 60
            CALL BACALG(B,S)
            SSET(B)=1
  60  CONTINUE
      DO 62 N=1,NACTS
           IF(D(B).EQ.0) GO TO 62
           DO 61 C=1,NCYC
                S2(B,C)=S(B,C)
 61        CONTINUE
 62  CONTINUE
     CALL SHIFT(S2,ORDER,FLAG)
     DO 65 B=1,NACTS
           IF(FLAG(B).EQ.0.OR.FLAG(P).EQ.3) GO TO 65
           CALL LOWBND(B,S2,LOW2(B),LENGT2(B))
           IF(TYPE.EQ.0.OR.TYPE.EQ.2)
    *           ICL=ICL+ICOSTF(B)*(LOW2(B)-LOW1(B))
           IF(TYPE.EQ.1.OR.TYPE.EQ.2)
    *           ICL=ICL+ICOSTV(B)*(LENGT2(B)-LENGT1(B))
           IF(TYPE.EQ.3)
    *           ICL=ICL+(LOW2(B)-LOW1(B))
```

```
            IF(TYPE.EQ.4)
     *           TCL=TCL+(LENGT2(B)**2)-(LENGT1(B)**2)
  65  CONTINUE
C***RESTORE ESS OF ALL ACTIVITIES
      DO 90 B=1,NACTS
            IF(FLAG(B)) 70,90,70
  70        DO 80 CYC=1,NCYC
                  HOLD=ESS(B,CYC)
                  ESS(B,CYC)=TESS(B,CYC)
                  TESS(B,CYC)=HOLD
  80        CONTINUE
  90  CONTINUE
C***
C***COMPUTING ICM
C***FIND BS FOR EACH ACTIVITY IN R NOT IN M'
      DO 100 N=NACTS,1,-1
            B=ORDER(N)
            IF(FLAG(B).EQ.0.OR.FLAG(B).EQ.5.OR.D(B).EQ.0)
     *           GO TO 100
            CALL BACALG(B,S1)
C***CALCULATE LOWER BOUND ON INTERRUPTIONS
            CALL LOWBND(B,S1,LOW1(B),LENGT1(B))
 100  CONTINUE
C***FIND FS FOR ACTIVITIES IN M'
      DO 130 N=NACTS,1,-1
            B=ORDER(N)
            IF(FLAG(B)-5) 130,110,130
 110        CALL FORALG(B,S1)
C***SET LSS OF B=FS
            DO 120 CYC=1,NCYC
                  TLSS(B,CYC)=LSS(B,CYC)
                  LSS(B,CYC)=S1(B,CYC)
 120        CONTINUE
 130  CONTINUE
C***CALCULATE ICM
      ICM=0
      DO 150 N=NACTS,1,-1
            B=ORDER(N)
            IF(FLAG(B).EQ.0.OR.FLAG(B).EQ.5) GO TO 150
C***SAVE LSS IN TEMPORARY STORAGE
            DO 140 CYC=1,NCYC
                  TLSS(B,CYC)=LSS(B,CYC)
 140        CONTINUE
C***FIND LSS OF ACTIVITIES IN R IF ACTIVITIES IN M'
C*** HAVE LSS=FS
            CALL CYCLSS(B,MAXD)
C***FIND NUMBER OF INTERRUPTIONS IN BS SCHEDULE
            IF(D(B).EQ.0) GO TO 150
            CALL BACALG(B,S1)
            SSET(B)=1
```

```
 150  CONTINUE
      DO 152 B=1,NACTS
           IF(D(B).EQ.0) GO TO 152
           DO 151 C=1,NCYC
                S2(B,C)=S1(B,C)
 151       CONTINUE
 152  CONTINUE
      CALL SHIFT(S2,ORDER,FLAG)
      DO 155 B=1,NACTS
           IF(FLAG(B).EQ.0.OR.FLAG(B).EQ.5) GO TO 155
           CALL LOWBND(B,S2,LOW2(B),LENGT2(B))
           IF(TYPE.EQ.0.OR.TYPE.EQ.2)
     *          TCM=TCM+ICOSTF(B)*(LOW2(B)-LOW1(B))
           IF(TYPE.EQ.1.OR.TYPE.EQ.2)
     *          TCM=TCM+ICOSTV(B)*(LENGT2(B)-LENGT1(B))
           IF(TYPE.EQ.3)
     *          TCM=TCM+LOW2(B)-LOW1(B)
           IF(TYPE.EQ.4)
     *          TCM=TCM+(LENGT2(B)**2)-(LENGT1(B)**2)
 155  CONTINUE
C***FIND THE SMALLER OF TCL AND TCM
      IF(TCL-TCM) 160,160,270
C***SCHEDULE ACTIVITIES IN L'TO BS
 160  SCHCNT=SCHCNT+COUNT(2)
      COUNT(2)=0
      DO 260 N=1,NACTS
           B=ORDER(N)
           IF(FLAG(B)) 170,260,170
 170       IF(FLAG(B)-3) 220,180,200
 180       FLAG(B)=0
                SSET(B)=1
           CALL UPCNT(B,DUM)
           IF(IND1.EQ.1) CALL TRACIT(180,'L''',S',B,S)
           DO 190 CYC=1,NCYC
                FSS(B,CYC)=S(B,CYC)
 190       CONTINUE
C***RESTORE LSS OF ALL ACTIVITIES
 200       IF(FLAG(B)-5) 220,210,220
 210       COUNT(4)=COUNT(4)-1
           FLAG(B)=1
 220       DO 230 CYC=1,NCYC
                LSS(B,CYC)=TLSS(B,CYC)
 230       CONTINUE
C***UPDATE ESS TIMES
           DO 250 CYC=1,NCYC
                IF(FLAG(B)) 240,250,240
 240            FSS(B,CYC)=TESS(B,CYC)
 250       CONTINUE
 260  CONTINUE
```

```
C***UPDATE THE SETS
      CALL SET
      RETURN 4
C***SCHEDULE ACTIVITIES IN M'TO FS
 270  SCHCNT=SCHCNT+COUNT(4)
      COUNT(4)=0
      DO 310 N=NACTS,1,-1
           B=ORDER(N)
           IF(FLAG(B)) 280,310,280
 280       IF(FLAG(B)-5) 310,290,310
 290       DO 300 CYC=1,NCYC
                S(B,CYC)=S1(B,CYC)
 300       CONTINUE
           FLAG(B)=0
                     SSET(B)=2
           CALL UPCNT(B,DUM)
           IF(IND1.EQ.1) CALL TRACIT(300,'M''',S',B,S)
 310  CONTINUE
C***UPDATE THE SETS
      CALL SET
      RETURN 5
      END
C*************************************************************
      SUBROUTINE SHIFT(S,ORDER,FLAG)
      IMPLICIT INTEGER(A-Z)
C***
C***VARIABLES
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      COMMON /NINE/SSET(100)
      DIMENSION S(100,50),ORDER(100),FLAG(100)
C***
C***INITIALIZE ARRAYS FOR DUMMIES
      CALL INIT1(S,ORDER)
C***RIGHT SHIFT ACTIVITIES SCHEDULED OUT OF L
      N=NACTS-1
      DO 50 A=N,2,-1
           B=ORDER(A)
           IF(SSET(B).NE.1) GO TO 50
           C=1
           X=1
           E=NCYC-1
   5       SLACK=1000000
           DO 10 F=X,E
                IF((S(B,F+1)-S(B,F)-D(B)).GT.0) GO TO 15
                C=C+1
  10       CONTINUE
  15       IF(C.EQ.NCYC.AND.X.NE.1) GO TO 50
           BB=B+1
```

```
            DO 30 F=B,NACTS
                 IF(I(F).NE.J(B)) GO TO 30
                 DO 20 G=X,C
                      SLACK=MIN(SLACK,(S(F,G)-S(B,G)-D(B)))
   20            CONTINUE
   30       CONTINUE
            IF(C.LT.NCYC)
     *           SLACK=MIN(SLACK,(S(F,C+1)-S(B,C)-D(B)))
            IF(SLACK.EQ.0) GO TO 45
            DO 40 F=X,C
                 S(B,F)=S(B,F)+SLACK
   40       CONTINUE
            CALL INIT1(S,ORDER)
   45       C=C+1
            X=C
            IF(C.GE.NCYC) GO TO 50
            GO TO 5
   50  CONTINUE
C***LEFT SHIFT FIRST CYCLES
       CALL INIT2(S,ORDER)
       DO 100 A=2,N
            B=ORDER(A)
            IF(SSET(B).NE.2) GO TO 100
            CALL LOWBND(B,S,LOW,DUM)
            IF(LOW.EQ.0) GO TO 100
            C=NCYC
            X=NCYC
   55       SLACK=1000000
            DO 60 F=X,2,-1
                 IF((S(B,F)-S(B,F-1)-D(B)).GT.0) GO TO 65
                 C=C-1
   60       CONTINUE
   65       BB=B-1
            DO 80 F=1,BB
                 IF(J(F).NE.I(B)) GO TO 80
                 DO 70 G=X,C,-1
                      SLACK=MIN(SLACK,(S(B,G)-S(F,G)-D(F)))
   70            CONTINUE
   80       CONTINUE
            IF(C.GT.1) SLACK=MIN(SLACK,(S(B,C)-S(B,C-1)-D(B)))
            IF(SLACK.EQ.0) GO TO 95
            DO 90 F=X,C,-1
                 S(B,F)=S(B,F)-SLACK
   90       CONTINUE
            CALL INIT2(S,ORDER)
   95  C=C-1
            X=C
            IF(C.LE.1) GO TO 100
            GO TO 55
  100  CONTINUE
```

```
      RETURN
      END
C*************************************************************
      SUBROUTINE INIT1(S,ORDER)
      IMPLICIT INTEGER (A-Z)
C***
C**VARIABLES
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      DIMENSION S(100,50),ORDER(100)
      N=NACTS-1
      DO 40 B=N,1,-1
           IF(D(B).NE.0) GO TO 40
           DO 10 C=1,NCYC
                S(B,C)=1000000
10         CONTINUE
           BB=B+1
           DO 30 C=BB,NACTS
                IF(I(C).NE.J(B)) GO TO 30
                DO 20 E=1,NCYC
                     S(B,E)=MIN(S(B,E),S(C,E))
20              CONTINUE
30         CONTINUE
40    CONTINUE
      RETURN
      END
C*************************************************************
      SUBROUTINE INIT2(S,ORDER)
      IMPLICIT INTEGER(A-Z)
C***
C**VARIABLES
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      DIMENSION S(100,50),ORDER(100)
C***
       DO 7 A=1,NACTS
            IF(D(A).NE.0) GO TO 7
            DO 5 B=1,NCYC
                 S(A,B)=0
5           CONTINUE
7     CONTINUE
      DO 30 B=2,NACTS
            IF(D(B).NE.0) GO TO 30
            BB=B-1
            DO 20 C=1,BB
                 IF(J(C).NE.I(B)) GO TO 20
                 DO 10 E=1,NCYC
                      S(B,E)=MAX(S(B,E),(S(C,E)+D(C)))
```

```
   10             CONTINUE
   20         CONTINUE
   30   CONTINUE
        RETURN
        END
C*************************************************************
        SUBROUTINE RANGEN(SEED,NSEED,IND4,IND5)
        IMPLICIT INTEGER (A-Z)
C***VARIABLES
        REAL RAND,RAND2
        DIMENSION TIMES(50,50)
        COMMON /ONE/I(100),J(100)
        COMMON /TWO/NACTS,NCYC
        COMMON /THREE/D(100)
        COMMON /FOUR/STAR(100),STRCNT
        COMMON /SIX/PCNT(50),SCNT(50),DUM1(100),DUM2(5)
        COMMON /SEVEN/ICOSTF(100),ICOSTV(100),DESC(100,4),
       *DCOST
        NAMELIST /OUTPUT/NACTS,NCYC,NODES,STRCNT
C***
        DO 1 A=1,100
             STAR(A)=' '
    1   CONTINUE
C***GENERATE A RANDOM NUMBER OF NODES: 20-50
        STRCNT=0
        SEEDX=SEED
        CALL GEN(RAND,SEEDX,NSEED)
        NODES=1+50*RAND
        DO 5 A=1,NODES
             PCNT(A)=0
             SCNT(A)=0
    5   CONTINUE
        MINACT=1.25*NODES
C***GENERATE A RANDOM NUMBER OF ACTIVITIES: <100
        CALL GEN(RAND,SEEDX,NSEED)
        MAXACT=MIN(100,((NODES*(NODES-1))/2))
        NACTS=MINACT+RAND*(MAXACT-MINACT)
C***ESTABLISH ACTIVITIES AND GENERATE DURATION TIMES
        Z=NODES-1
        DO 20 A=1,Z
             Y=A+1
             DO 10 B=Y,NODES
                  TIMES(A,B)=' '
   10        CONTINUE
   20   CONTINUE
C***SCHEDULE AND ACTIVITY ON EVERY ROW
        COUNT=Z
        DO 30 A=1,Z
             CALL GEN(RAND,SEEDX,NSEED)
             COL=1+A+RAND*(NODES-A)
             IF(COL.GT.NODES) COL=NODES
```

```
            CALL GEN(RAND,SEEDX,NSEED)
            TIMES(A,COL)=RAND*21
   30   CONTINUE
C***CHECK COLUMNS, ENTER ACTIVITIES IN EMPTY COLUMNS
        DO 50 A=NODES,2,-1
            Y=A-1
            DO 40 B=1,Y
                IF(TIMES(B,A).NE.' ') GO TO 50
   40       CONTINUE
            CALL GEN(RAND,SEEDX,NSEED)
            ROW=1+RAND*Y
            IF(ROW.EQ.A) ROW=A-1
            CALL GEN(RAND,SEEDX,NSEED)
            TIMES(ROW,A)=RAND*21
            COUNT=COUNT+1
   50   CONTINUE
C***FILL REMAINING CELLS AT RANDOM
   60   IF(COUNT.EQ.NACTS) GO TO 80
   70   CALL GEN(RAND,SEEDX,NSEED)
        ROW=1+RAND*NODES
        IF(ROW.GT.NODES) ROW=NODES
        CALL GEN(RAND,SEEDX,NSEED)
        COL=ROW+1+RAND*(NODES-ROW)
        IF(COL.GT.NODES) COL=NODES
        IF(TIMES(ROW,COL).NE.' ') GO TO 70
        CALL GEN(RAND,SEEDX,NSEED)
        TIMES(ROW,COL)=RAND*21
        COUNT=COUNT+1
        GO TO 60
C***PUT DATA INTO MAIN ARRAYS
   80   N=1
        DO 100 A=1,Z
            C=A+1
            DO 90 B=C,NODES
                IF(TIMES(A,B).EQ.' ') GO TO 90
                I(N)=A
                J(N)=B
                SCNT(A)=SCNT(A)+1
                PCNT(B)=PCNT(B)+1
                D(N)=TIMES(A,B)
                CALL GEN(RAND,SEEDX,NSEED)
                IF(D(N).NE.0) ICOSTF(N)=RAND*10+1
                CALL GEN(RAND,SEEDX,NSEED)
                IF(D(N).NE.0) ICOSTV(N)=RAND*10+1
                N=N+1
   90       CONTINUE
  100   CONTINUE
C***DETERMINE THE NUMBER OF CYCLES
        CALL GEN(RAND,SEEDX,NSEED)
        NCYC=3+RAND*19
C***SELECT ACTIVITIES TO BE CONTINUOUS
```

```
      IF(IND5.NE.1) GO TO 106
      CALL GEN(RAND2,SEEDX,NSEED)
      DO 105 N=1,NACTS
           CALL GEN(RAND,SEEDX,NSEED)
           IF(RAND.GE.RAND2.OR.D(N).EQ.0) GO TO 105
           STRCNT=STRCNT+1
           STAR(N)='*'
  105 CONTINUE
  106 IF(IND4.NE.1) RETURN
      WRITE(6,OUTPUT)
      WRITE(6,5000)
      DO 107 N=1,NACTS
           WRITE(6,5001)N,STAR(N),I(N),J(N),D(N),ICOST(N)
  107 CONTINUE
 5000 FORMAT(1X,'NO',4X,'I',2X,'J',1X,'TIME',1X,'COST')
 5001 FORMAT(1X,I2,1X,A1,1X,I2,1X,I2,2X,I2,3X,I2)
      RETURN
C*****************!*******
      SUBROUTINE GEN(X,Y,KD)
      REAL X,E
      IF(Y.EQ.0) GO TO 110
      KA=Y
      KB=5**7
      KD=KA
      Y=0
  110 KA=KD
      KC=KA*KB
      KD=MOD(KC,2**17)
      E=KD
      X=E/(2.0**17)
      RETURN
      END
C*************************************************************
      SUBROUTINE TRACIT(LABEL,TYPE1,TYPE2,N,S)
      IMPLICIT INTEGER(A-Z)
C**VARIABLES
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      COMMON /SIX/PCNT(50),SCNT(50),FLAG(100),COUNT(5)
      DIMENSION S(100,50),SETS(6),LINE(100)
      DATA SETS/'R','L','L-','M','M-','K'/
C**
 5000 FORMAT( )
      WRITE(6,5001)LABEL,N,I(N),J(N),D(N),TYPE1,TYPE2
 5001 FORMAT(1H0,4X,14HPASSING LABEL ,I5,/5X,9HACTIVITY ,I3,
     */,5X,'I=',I2,' J=',I2,' D=',I2,/5X,
     *18H DELETED FROM SET ,A3,18H AND ADDED TO SET ,A3,
     *47HTHE SCHEDULE FOR THE ACTIVITY IN EACH CYCLE IS:)
      WRITE(6,5000)(S(N,K),K=1,NCYC)
```

```
      WRITE(6,5004)(COUNT(K),K=1,5)
5004  FORMAT(5X,19HTHE SET COUNTS ARE ,7(I2,2X))
      MAXJ=J(NACTS)
      WRITE(6,5005)(PCNT(K),K=1,MAXJ)
5005  FORMAT(5X,'PCNT ',25(I2,1X))
      WRITE(6,5006)(SCNT(K),K=1,MAXJ)
5006  FORMAT(5X,'SCNT ',25(I2,1X))
      DO 30 A=1,NACTS
           IF(FLAG(A)) 10,10,20
   10      LINE(A)='S'
           GO TO 30
   20      B=FLAG(A)
           LINE(A)=SETS(B)
   30 CONTINUE
      WRITE(6,5003)
5003  FORMAT(5X,11HTHE SET IS:)
      INDEX=1
      TIMES=NACTS/10
      IF(MOD(NACTS,10).NE.0) TIMES=TIMES+1
      DO 40 A=1,TIMES
           INDEX2=MIN(NACTS,INDEX+9)
           WRITE(6,5002)(B,LINE(B)),B=INDEX,INDEX2)
5002       FORMAT(10(5X,I3,',',A3))
           INDEX=INDEX+10
   40 CONTINUE
      RETURN
      END
C*************************************************************
      SUBROUTINE TIMWRT(IND3,MAXD)
      IMPLICIT INTEGER(A-Z)
C***VARIABLES
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /FIVE/ESS(100,50),LSS(100,50)
C***
      B=1
   10 WRITE(6,5000)(K,K=1,NCYC)
5000  FORMAT(1H1,8HACTIVITY,50X,5HCYCLE,/,1X,
     *8HNO   I   J,3X,29(2X,I2))
      DO 30 A=B,NACTS
           WRITE(6,5001)A,I(A),J(A),(ESS(A,M),M=1,NCYC),
     *     (LSS(A,N),N=1,NCYC)
5001       FORMAT(3(1X,I2),1X,3HESS,29(1X,I3),/,
     *     10X,3HLSS,29(1X,I3))
           IF(A.EQ.B+39.AND.A.LT.NACTS) GO TO 20
           GO TO 30
   20      B=B+40
           GO TO 10
   30 CONTINUE
      IF(IND3.EQ.1) CALL CYCCHT(ESS,MAXD)
      IF(IND3.EQ.1) CALL CYCCHT(LSS,MAXD)
```

```
      RETURN
      END
C*********************************************************
      SUBROUTINE CHECK(S)
      IMPLICIT INTEGER(A-Z)
C***VARIABLES
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      DIMENSION S(100,50)
C***
      DO 50 N=1,NACTS
           IF(I(N).EQ.1.OR.D(N).EQ.0) GO TO 50
           NP=N-1
           DO 40 M=1,NP
                IF(D(M).EQ.0) GO TO 40
                IF(I(N)-J(M)) 40,10,40
  10            DO 20 CYC=1,NCYC
                     IF((S(M,CYC)+D(M)).GT.S(N,CYC)) GO TO 3
  20            CONTINUE
                GO TO 40
  30            WRITE(6,5000)M,I(M),J(M),N,I(N),J(N)
5000  FORMAT(1X,9HACTIVITY ,I2,5H (I= ,I2,1H,,4H J= ,I2,1H),
     *5H AND ,9HACTIVITY ,I2,5H (I= ,I2,1H,,4H J= ,I2,1H),
     *26H HAVE INFEASIBLE SCHEDULES)
  40        CONTINUE
  50  CONTINUE
      RETURN
      END
C*********************************************************
      SUBROUTINE CYCLST(S,MIND)
      IMPLICIT INTEGER(A-Z)
C***VARIABLES
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      COMMON /FOUR/STAR(100),STRCNT
      COMMON /SEVEN/ICOSTF(100),ICOSTV(100),DESC(100,4),
     *DCOST
      COMMON /EIGHT/PAGE
      DIMENSION S(100,50)
C***
      TCOST=0
      INDEX1=1
      INDEX2=1
  10  WRITE(6,5000)
5000  FORMAT(1H1,7(/),51X,31HMULTIPLE CYCLE PROJECT SCHEDULE,
     *//,21X,8HACTIVITY,5X,5HNODES,
     *,13X,8HACTIVITY,37X,5HSTART,5X,6HFINISH,/,24X,2HNO,9X,
     *4HI  J,12X,11HDESCRIPTION,12X,5HCYCLE,5X,8HDURATION,5X,
     *4HTIME,7X,4HTIME,/)
```

```
      LINENO=0
      DO 80 A=INDEX1,NACTS
            DO 70 B=INDEX2,NCYC
                  IF(D(A).EQ.0) GO TO 80
                  FT=S(A,B)+D(A)
                  IF(B.EQ.1.OR.LINENO.EQ.0) GO TO 20
                  GO TO 30
   20             WRITE(6,5001)A,STAR(A),I(A),J(A),
     *            (DESC(A,K),K=1,4),B,D(A),S(A,B),FT
 5001             FORMAT(/,23X,I3,1X,A1,5X,2(1X,I2),
     *            6X,4(A6),6X,2(I3,8X),2(I4,7X))
                  LINENO=LINENO+2
                  IF(LINENO-35) 70,40,40
   30             WRITE(6,5002)B,D(A),S(A,B),FT
 5002             FORMAT(75X,2(I3,8X),2(I4,7X))
                  LINENO=LINENO+1
                  IF(LINENO-35) 70,40,40
   40             IF(A.EQ.NACTS.AND.B.EQ.NCYC) GO TO 90
                  IF(B-NCYC) 50,60,60
   50             INDEX1=A
                  INDEX2=B+1
                  GO TO 10
   60             INDEX1=A+1
                  INDEX2=1
                  GO TO 10
   70       CONTINUE
            INDEX2=1
   80    CONTINUE
   90    INDEX1=1
  100    WRITE(6,5003)
 5003    FORMAT(1H1,6(/),50X,22HMULTIPLE CYCLE PROJECT,
     *//,55X,12HCOST SUMMARY,//,16X,5HNODES,10X,
     *8HACTIVITY,14X,13HINTERRUPTIONS,6X,
     *10HUNIT COSTS,3X,11HTOTAL COSTS,/,17X,4HI  J,
     *9X,11HDESCRIPTION,12X,13HNUMBER LENGTH,
     *2(4X,14HFIXED VARIABLE,1X),/)
      LINENO=0
      DO 130 A=INDEX1,NACTS
            IF(D(A).EQ.0) GO TO 130
            CALL LOWBND(A,S,LOW,LENGTH)
            TLENG=TLENG+LENGTH
            TLOW=TLOW+LOW
            FCOST=LOW*ICOSTF(A)
            VCOST=LENGTH*ICOSTV(A)
            TFCOST=TFCOST+FCOST
            TVCOST=TVCOST+VCOST
            WRITE(6,5004)I(A),J(A),(DESC(A,K),K=1,4),LOW,
     *      LENGTH,ICOSTF(A),ICOSTV(A),FCOST,VCOST
 5004       FORMAT(16X,2(I2,1X),3X,4A6,6X,I2,4X,I4,6X,
     *      2(I3,4X),4X,I5,2X,I6)
            LINENO=LINENO+1
```

```
            IF(LINENO-35) 130,110,110
  110       IF(A-NACTS) 120,130,130
  120       INDEX1=A+1
            GO TO 100
  130       CONTINUE
      WRITE(6,5005)TLOW,TLENG,TFCOST,TVCOST
 5005 FORMAT(/53X,4H----,3X,5H-----,24X,5H-----,2X,
     *6H------,/,53X,I4,3X,I5,24X,I5,2X,I6)
      DELAY=DCOST*(S(NACTS,NCYC)-MIND)
      TCOST=TFCOST+TVCOST+DELAY
      WRITE(6,5006)DELAY,TCOST
 5006 FORMAT(/,16X,'DELAY COST = ',I6,'  TOTAL COST =',
     *I6)
      RETURN
      END
C*************************************************************
      SUBROUTINE CYCCNT(S,MAXD)
      IMPLICIT INTEGER(A-Z)
C***VARIABLES
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      COMMON /FOUR/STAR(100),STRCNT
      COMMON /FIVE/ESS(100,50),LSS(100,50)
      COMMON /SEVEN/ICOSTF(100),ICOSTV(100),DESC(100,4),
     *DCOST
      DIMENSION LINE(100),CYCCNT(100),S(100,50),NUMB(10)
      DATA NUMB/'0','1','2','3','4','5','6','7','8','9'/
C***
      DO 1 A=1,100
           CYCCNT(A)=1
    1 CONTINUE
      TIMES=MAXD/100
      IF(MOD(MAXD,100).NE.0) TIMES=TIMES+1
      DO 140 A=1,TIMES
           BEGING=(A-1)*100
           ENDNG=(A*100)-1
           INDX=1
   10      LINENO=0
           WRITE(6,4999)
 4999      FORMAT(1H1,////////,16X,3HSC,47X,10HTIME UNITS)
           IF(A-1) 20,20,40
   20      DO 30 B=1,100
                LINE(B)=' '
   30      CONTINUE
           WRITE(6,5000)(LINE(K),K=1,100)
 5000      FORMAT(16X,2HTY,1X,100A1)
           WRITE(6,5001)(LINE(K),K=1,100)
 5001      FORMAT(16X,2HAC,1X,100A1)
           WRITE(6,5002)
 5002      FORMAT(10X,5HNODES,1X,2HPL,1X,10(1H ),10(1H1),
```

```
     +      10(1H2),10(1H3),10(1H4),10(1H5),10(1H6),10(1H7),
     +      10(1H8),10(1H9))
            GO TO 60
   40       IF(A.GT.10) GO TO 42
            DO 41 B=1,100
                 LINE(B)=' '
   41       CONTINUE
            WRITE(6,5000)(LINE(K),K=1,100)
            GO TO 44
   42       LOW=(A-1)/10
            DO 43 B=1,100
                 LINE(B)=NUMB(LOW+1)
   43       CONTINUE
            WRITE(6,5000)(LINE(K),K=1,100)
   44       C=A-LOW*10
            DO 50 B=1,100
                 LINE(B)=NUMB(C)
   50       CONTINUE
            WRITE(6,5001)(LINE(K),K=1,100)
            WRITE(6,5003)
 5003       FORMAT(10X,5HNODES,1X,2HPL,1X,10(1H0),10(1H1),
     +      10(1H2),10(1H3),10(1H4),10(1H5),10(1H6),10(1H7),
     +      10(1H8),10(1H9))
   60       WRITE(6,5004)
 5004       FORMAT(11X,4HI  J,1X,2HTF,1X,10(10H0123456789),/)
            DO 130 B=INDX,NACTS
                 IF(D(B).EQ.0) GO TO 130
                 DO 70 BB=1,100,2
                      LINE(BB)='.'
                      BBB=BB+1
                      LINE(BBB)=' '
   70            CONTINUE
                 II=I(B)
                 JJ=J(B)
                 DD=D(B)
                 IF(S(B,1).GT.ENDNG.OR.(S(B,NCYC)+D(B)-1).
     *           LT.BEGNG) GO TO 71
                 GO TO 72
   71            WRITE(6,5005)STAR(B),II,JJ,CYCCNT(B),
     *           (LINE(K),K=1,100)
                 LINENO=LINENO+1
                 GO TO 120
   72            CC=CYCCNT(B)
                 DO 100 C=CC,NCYC
                      DO 90 F=1,DD
                           SS=S(B,C)+F-1
                           IF(SS.LT.BEGNG) GO TO 90
                           IF(SS.GT.ENDNG) GO TO 80
                           IF(SS.LT.100) SSS=SS+1
                           IF(SS.GE.100)
     *                          SSS=MOD(SS,((A-1)*100))+1
```

```
                              CYC=MOD(C,10)+1
                              LINE(SSS)=NUMB(CYC)
                              GO TO 90
   80                         WRITE(6,5005)STAR(B),II,JJ,
     *                        CYCCNT(B),(LINE(K),K=1,100)
                              LINENO=LINENO+1
                              CYCCNT(B)=C
                              GO TO 120
   90                    CONTINUE
  100               CONTINUE
  110               WRITE(6,5005)STAR(B),II,JJ,CYCCNT(B),
     *              (LINE(K),K=1,100)
                    LINENO=LINENO+1
 5005               FORMAT(7X,A1,1X,3(1X,I2),1X,100A1)
                    CYCCNT(B)=NCYC
  120               IF(B.EQ.NACTS) GO TO 130
                    IF(LINENO.NE.30) GO TO 130
                    INDX=B+1
                    GO TO 10
  130          CONTINUE
  140     CONTINUE
          VCOST=0
          FCOST=0
          PDCOST=0
          TCOST=0
          DO 150 N=1,NACTS
               IF(D(N).EQ.0) GO TO 150
               CALL LOWBND(N,S,LOW,LENGTH)
               FCOST=FCOST+LOW*ICOSTF(N)
               VCOST=VCOST+LENGTH*ICOSTV(N)
  150     CONTINUE
          PDCOST=DCOST*(MAXD-ESS(NACTS,NCYC)-D(NACTS))
          TCOST=FCOST+VCOST+PDCOST
          WRITE(6,5006)FCOST,VCOST,PDCOST,TCOST
 5006     FORMAT(/17X,'FIXED COST = ',I6,' DAILY COST = ',I6,
        *' DELAY COST = ',I6,' TOTAL COST = ',I6)
          RETURN
          END
```

```
$INPUT NACTS=33,NCYC=5,TYPE=2,LIST=1,CHART=1,$END
0102ACTIVITY 1-2                004002002
0203ACTIVITY 2-3                002003003
0304ACTIVITY 3-4                014001001
0405ACTIVITY 4-5                002004004
0506ACTIVITY 5-6                005001001
0507ACTIVITY 5-7                001003003
0510ACTIVITY 5-10               010001001
0511ACTIVITY 5-11               003005005
0512ACTIVITY 5-12               010003003
0514ACTIVITY 5-14               016002002
0608ACTIVITY 6-8                005002002
0709ACTIVITY 7-9                002002002
0810ACTIVITY 8-10               005002002
0814ACTIVITY 8-14               005002002
0911DUMMY 9-11                  000000000
0914DUMMY 9-14                  000000000
1012ACTIVITY 10-12              005003003
1115ACTIVITY 11-15              003002002
1213ACTIVITY 12-13              003004004
1314ACTIVITY 13-14              003003003
1415ACTIVITY 14-15              003004004
1516ACTIVITY 15-16              005001001
1517ACTIVITY 15-17              003003003
1617DUMMY 16-17                 000000000
1618DUMMY 16-18                 000000000
1619ACTIVITY 16-19              003002002
1719ACTIVITY 17-19              005002002
1819ACTIVITY 18-19              005001001
1920DUMMY 19-20                 000000000
1921ACTIVITY 19-21              005001001
2021ACTIVITY 20-21              005003003
2122ACTIVITY 21-22              003005005
2223ACTIVITY 22-23              004002002
```

MULTIPLE CYCLE PROJECT SCHEDULE

ACTIVITY NO	NODES I	NODES J	ACTIVITY DESCRIPTION	CYCLE	DURATION	START TIME	FINISH TIME
1	1	2	ACTIVITY 1-2	1	4	0	4
				2	4	4	8
				3	4	8	12
				4	4	12	16
				5	4	16	20
2	2	3	ACTIVITY 2-3	1	2	4	6
				2	2	14	16
				3	2	16	18
				4	2	18	20
				5	2	20	22
3	3	4	ACTIVITY 3-4	1	14	6	20
				2	14	20	34
				3	14	34	48
				4	14	48	62
				5	14	62	76
4	4	5	ACTIVITY 4-5	1	2	22	24
				2	2	38	40
				3	2	54	56
				4	2	63	65
				5	2	76	78
5	5	6	ACTIVITY 5-6	1	5	50	55
				2	5	55	60
				3	5	60	65
				4	5	65	70
				5	5	78	83
6	5	7	ACTIVITY 5-7	1	1	79	80
				2	1	80	81
				3	1	81	82
				4	1	82	83

MULTIPLE CYCLE PROJECT SCHEDULE

ACTIVITY NO	NODES I	J	ACTIVITY DESCRIPTION	CYCLE	DURATION	START TIME	FINISH TIME
6	5	7	ACTIVITY 5-7	5	1	83	84
7	5	10	ACTIVITY 5-10	1	10	40	50
				2	10	50	60
				3	10	60	70
				4	10	70	80
				5	10	80	90
8	5	11	ACTIVITY 5-11	1	3	79	82
				2	3	82	85
				3	3	85	88
				4	3	88	91
				5	3	91	94
9	5	12	ACTIVITY 5-12	1	10	45	55
				2	10	55	65
				3	10	65	75
				4	10	75	85
				5	10	85	95
10	5	14	ACTIVITY 5-14	1	16	24	40
				2	16	40	56
				3	16	56	72
				4	16	72	88
				5	16	88	104
11	6	8	ACTIVITY 6-8	1	5	55	60
				2	5	60	65
				3	5	65	70
				4	5	70	75
				5	5	83	88
12	7	9	ACTIVITY 7-9	1	2	80	82
				2	2	82	84

MULTIPLE CYCLE PROJECT SCHEDULE

ACTIVITY NO	NODES I	J	ACTIVITY DESCRIPTION	CYCLE	DURATION	START TIME	FINISH TIME
12	7	9	ACTIVITY 7-9	3	2	84	86
				4	2	86	88
				5	2	88	90
13	8	10	ACTIVITY 8-10	1	5	60	65
				2	5	65	70
				3	5	70	75
				4	5	75	80
				5	5	88	93
14	8	14	ACTIVITY 8-14	1	5	71	76
				2	5	76	81
				3	5	81	86
				4	5	86	91
				5	5	91	96
17	10	12	ACTIVITY 10-12	1	5	65	70
				2	5	70	75
				3	5	75	80
				4	5	80	85
				5	5	93	98
18	11	15	ACTIVITY 11-15	1	3	82	85
				2	3	85	88
				3	3	88	91
				4	3	91	94
				5	3	94	97
19	12	13	ACTIVITY 12-13	1	3	76	79
				2	3	79	82
				3	3	82	85
				4	3	85	88
				5	3	98	101
20	13	14	ACTIVITY 13-14	1	3	79	82

MULTIPLE CYCLE PROJECT SCHEDULE

ACTIVITY NO	NODES I	NODES J	ACTIVITY DESCRIPTION	CYCLE	DURATION	START TIME	FINISH TIME
20	13	14	ACTIVITY 13-14	2	3	82	85
				3	3	85	88
				4	3	88	91
				5	3	101	104
21	14	15	ACTIVITY 14-15	1	3	82	85
				2	3	85	88
				3	3	88	91
				4	3	91	94
				5	3	104	107
22	15	16	ACTIVITY 15-16	1	5	85	90
				2	5	90	95
				3	5	95	100
				4	5	100	105
				5	5	107	112
23	15	17	ACTIVITY 15-17	1	3	89	92
				2	3	92	95
				3	3	95	98
				4	3	98	101
				5	3	107	110
26	16	19	ACTIVITY 16-19	1	3	94	97
				2	3	99	102
				3	3	102	105
				4	3	105	108
				5	3	112	115
27	17	19	ACTIVITY 17-19	1	5	92	97
				2	5	97	102
				3	5	102	107
				4	5	107	112
				5	5	112	117

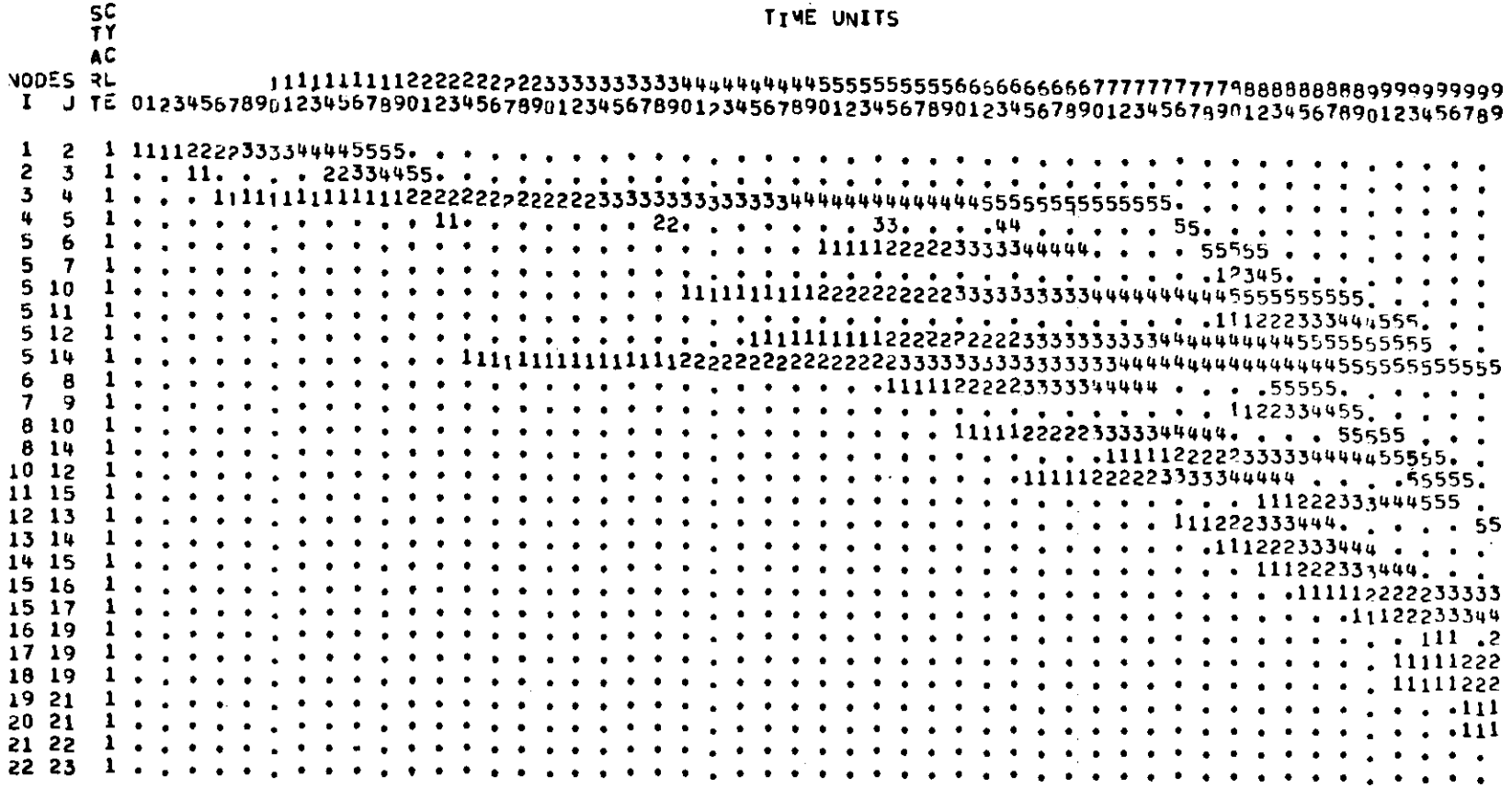
MULTIPLE CYCLE PROJECT SCHEDULE

ACTIVITY NO	NODES I	NODES J	ACTIVITY DESCRIPTION	CYCLE	DURATION	START TIME	FINISH TIME
28	18	19	ACTIVITY 18-19	1	5	92	97
				2	5	97	102
				3	5	102	107
				4	5	107	112
				5	5	112	117
30	19	21	ACTIVITY 19-21	1	5	97	102
				2	5	102	107
				3	5	107	112
				4	5	112	117
				5	5	117	122
31	20	21	ACTIVITY 20-21	1	5	97	102
				2	5	102	107
				3	5	107	112
				4	5	112	117
				5	5	117	122
32	21	22	ACTIVITY 21-22	1	3	106	109
				2	3	109	112
				3	3	112	115
				4	3	117	120
				5	3	122	125
33	22	23	ACTIVITY 22-23	1	4	109	113
				2	4	113	117
				3	4	117	121
				4	4	121	125
				5	4	125	129

MULTIPLE CYCLE PROJECT
COST SUMMARY

NODES I	NODES J	ACTIVITY DESCRIPTION	INTERRUPTIONS NUMBER	INTERRUPTIONS LENGTH	UNIT COSTS FIXED	UNIT COSTS VARIABLE	TOTAL COSTS FIXED	TOTAL COSTS VARIABLE
1	2	ACTIVITY 1-2	0	0	2	2	0	0
2	3	ACTIVITY 2-3	1	8	3	3	3	24
3	4	ACTIVITY 3-4	0	0	1	1	0	0
4	5	ACTIVITY 4-5	4	46	4	4	16	184
5	6	ACTIVITY 5-6	1	8	1	1	1	8
5	7	ACTIVITY 5-7	0	0	3	3	0	0
5	10	ACTIVITY 5-10	0	0	1	1	0	0
5	11	ACTIVITY 5-11	0	0	5	5	0	0
5	12	ACTIVITY 5-12	0	0	3	3	0	0
5	14	ACTIVITY 5-14	0	0	2	2	0	0
6	8	ACTIVITY 6-8	1	8	2	2	2	16
7	9	ACTIVITY 7-9	0	0	2	2	0	0
8	10	ACTIVITY 8-10	1	8	2	2	2	16
8	14	ACTIVITY 8-14	0	0	2	2	0	0
10	12	ACTIVITY 10-12	1	8	3	3	3	24
11	15	ACTIVITY 11-15	0	0	2	2	0	0
12	13	ACTIVITY 12-13	1	10	4	4	4	40
13	14	ACTIVITY 13-14	1	10	3	3	3	30
14	15	ACTIVITY 14-15	1	10	4	4	4	40
15	16	ACTIVITY 15-16	1	2	1	1	1	2
15	17	ACTIVITY 15-17	1	6	3	3	3	18
16	19	ACTIVITY 16-19	2	6	2	2	4	12
17	19	ACTIVITY 17-19	0	0	2	2	0	0
18	19	ACTIVITY 18-19	0	0	1	1	0	0
19	21	ACTIVITY 19-21	0	0	1	1	0	0
20	21	ACTIVITY 20-21	0	0	3	3	0	0
21	22	ACTIVITY 21-22	2	4	5	5	10	20
22	23	ACTIVITY 22-23	0	0	2	2	0	0
			18	134			56	434

DELAY COST = 0 TOTAL COST = 490


SC
TY
AC
TIME UNITS
NODES
I J
RL
TE
0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789

```
                                                    TIME UNITS
      SC
      TY
      AC 1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
NODES RE 0000000000111111111122222222223333333333444444444455555555556666666677777777778888888888999999999
 I  J TE 0123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
 1  2  5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 2  3  5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 3  4  5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 4  5  5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 5  6  5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 5  7  5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 5 10  5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 5 11  5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 5 12  5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 5 14  5 5555. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 6  8  5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 7  9  5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 8 10  5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
 8 14  5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
10 12  5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
11 15  5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
12 13  5 5 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
13 14  5 .555. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
14 15  5 . . 555 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
15 16  4 44444 .55555. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
15 17  4 4 . . .555. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
16 19  2 22333444. . 555 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
17 19  2 223333344444555555 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
18 19  2 22333334*44455555 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
19 21  1 11222223333344444455555. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
20 21  1 11222223333344444455555. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
21 22  1 . . . 111222333 .444. 555 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
22 23  1 . . . . .1111222333344445555 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

      FIXED COST =     56 DAILY COST =    434 DELAY COST =      0 TOTAL COST =    490
```

APPENDIX C

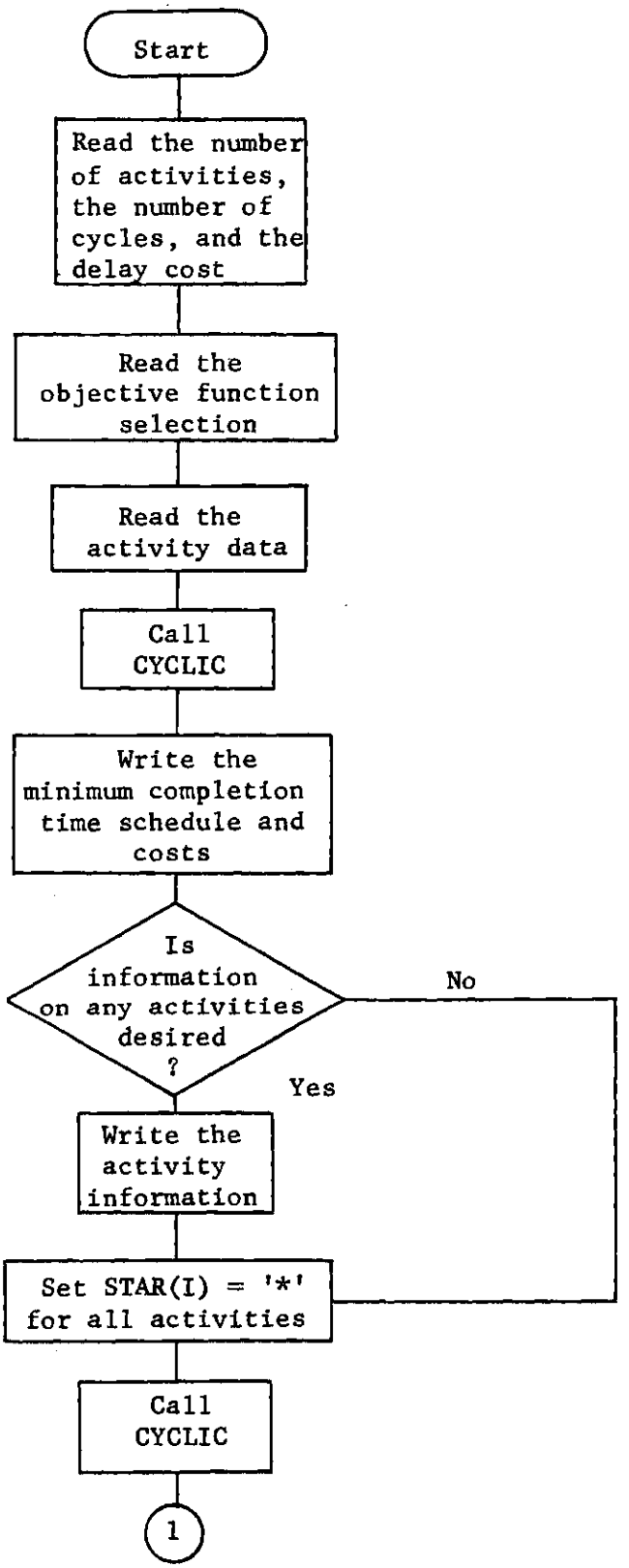
DOCUMENTATION OF THE DEMAND MODE SCHEDULING PROGRAM

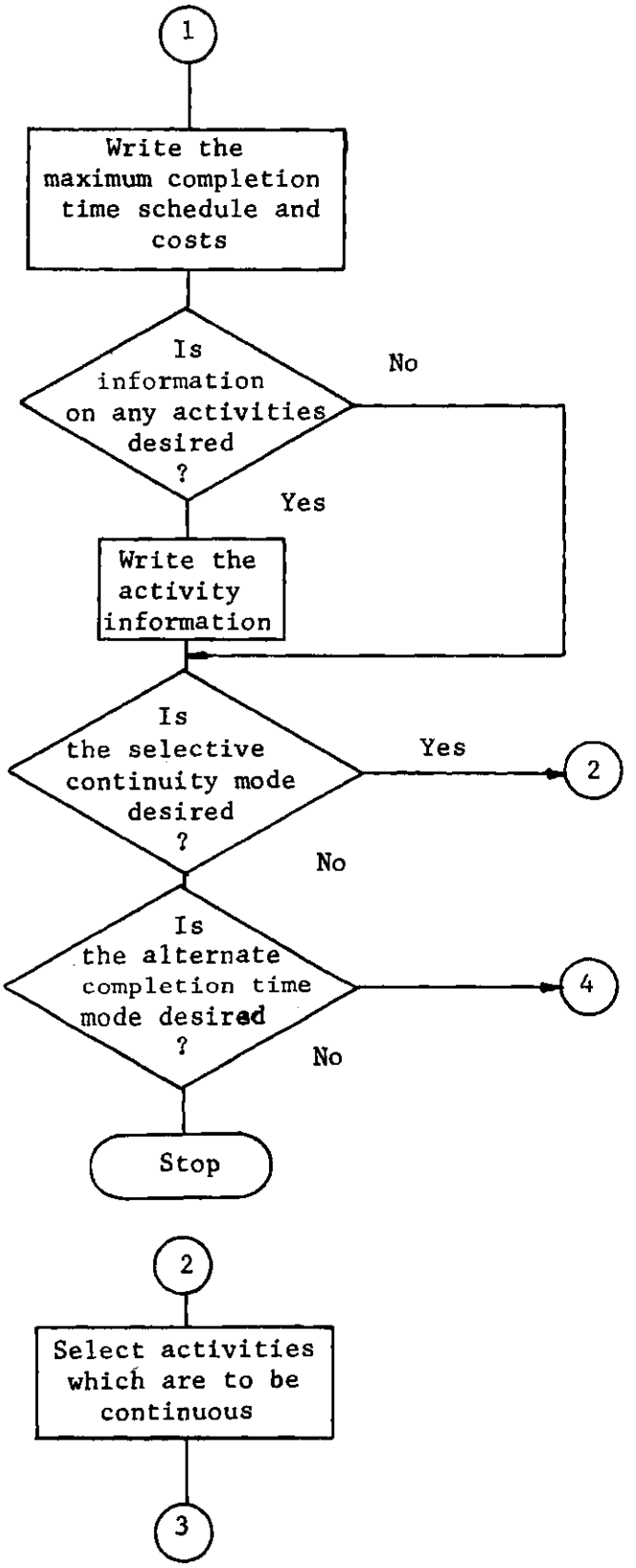
The demand mode scheduling program is very similar to the batch mode program in that it uses most of the scheduling subroutines that were employed in CPSS. All of the major variables are the same. The main program has several additional variables which are used as indicators, indices, or temporary storage areas. These may be easily determined from the context of the source listing of the program.

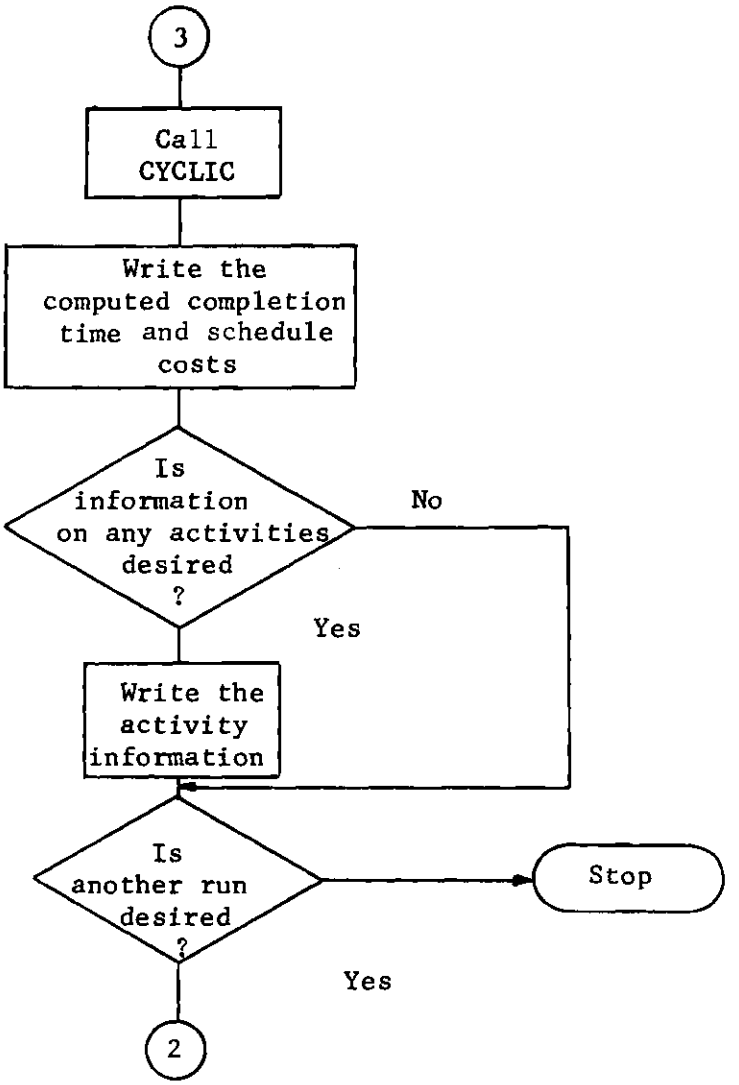
The only subroutines which are specialized to the use of the demand program are the subroutines WRITE1, WRITE2, and WRITE3. These subroutines perform the specialized operations of printing a listing of all continuous activities in a project schedule, all non-continuous activities in a project schedule, and the schedule of any specified activity, respectively.

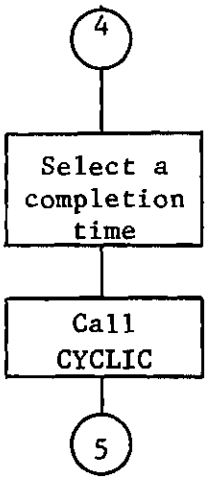
There are no specified input formats. All instructions for entering data are printed by the main program when it prompts the user for input. There are several varied output formats, all of which have been developed to interact with the user conversationally.

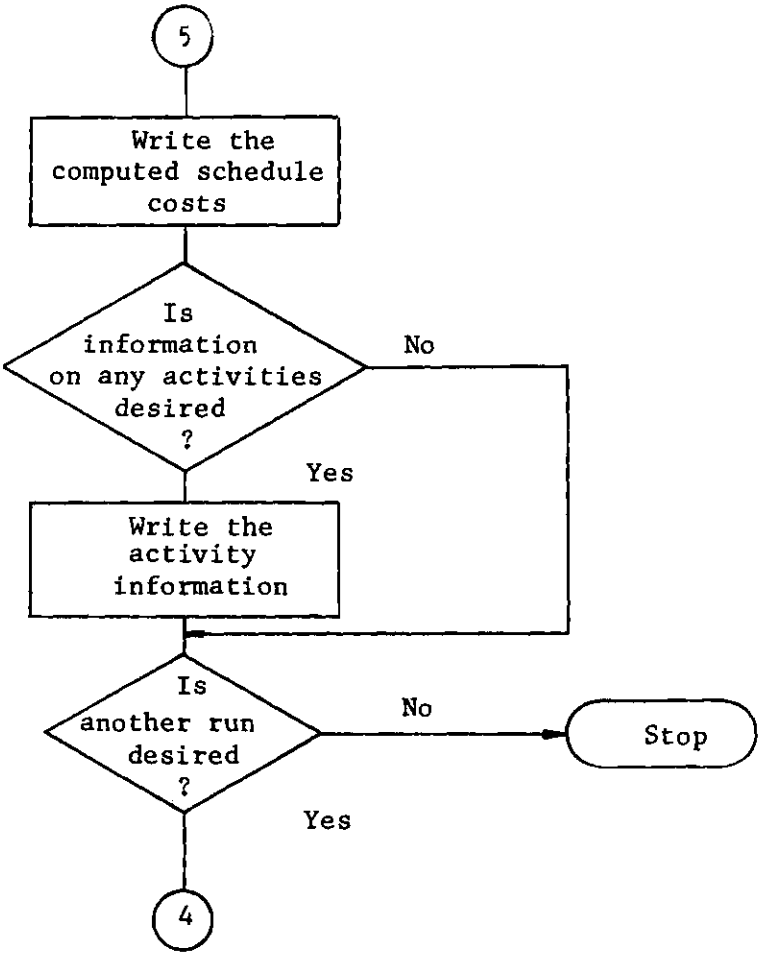
The following pages present, in order, a macro flowchart, a listing of the main program, and two sample runs of the demand mode program. The sample runs demonstrate very well the interactive features of this program and give some indication of their usefulness.


Start
Read the number of activities, the number of cycles, and the delay cost
Read the objective function selection
Read the activity data
Call CYCLIC
Write the minimum completion time schedule and costs
Is information on any activities desired ?
No
Yes
Write the activity information
Set STAR(I) = '*' for all activities
Call CYCLIC
1


1
Write the maximum completion time schedule and costs
Is information on any activities desired ?
No
Yes
Write the activity information
Is the selective continuity mode desired ?
Yes
2
No
Is the alternate completion time mode desired ?
4
No
Stop
2
Select activities which are to be continuous
3


3
Call CYCLIC
Write the computed completion time and schedule costs
Is information on any activities desired ?
No
Yes
Write the activity information
Is another run desired ?
Stop
Yes
2


4
Select a completion time
Call CYCLIC
5


5
Write the computed schedule costs
Is information on any activities desired ?
No
Yes
Write the activity information
Is another run desired ?
No
Stop
Yes
4

```
      IMPLICIT INTEGER (A-Z)
C***VARIABLES
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      COMMON /FOUR/STAR(100),STRCNT
      COMMON /FIVE/ESS(100,50),LSS(100,50)
      COMMON /SIX/PCNT(50),SCNT(50),FLAG(100),COUNT(5)
      COMMON /SEVEN/TCOSTF(100),ICOSTV(100),DESC(100,4),
     *DCOST
      DATA STAR/100*' '/
      DIMENSION S(100,50),LINE(100),LOW(100),PRCNT(50),
     *PSCNT(50)
C***FORMATS
1000  FORMAT( )
1001  FORMAT(A3)
5000  FORMAT(/1X,'THIS PROGRAM ALLOWS THE USER TO EXAMINE',
     *      /,1X,'RESULTING INTERRUPTION AND DELAY COSTS',
     *      /,1X,'FOR ALTERNATIVE SCHEDULES OF A PARALLEL',
     *      /,1X,'MULTI-CYCLE PROJECT.',
     *      /,1X,'DO YOU HAVE A PROBLEM TO RUN ?',
     *      /,1X,'(ENTER YES/NO)')
5001  FORMAT(/1X,'ENTER - IN INTEGER FORMAT AND SEPARATED',
     *      /,1X,'BY COMMAS - THE NUMBER OF ACTIVITIES,',
     *      /,1X,'THE NUMBER OF CYCLES IN THE PROJECT,',
     *      /,1X,'AND THE PER DAY PROJECT DELAY COST.')
5002  FORMAT(/1X,'ENTER - ONE LINE AT A TIME, IN INTEGER',
     *      /,1X,'FORMAT, AND SEPARATED BY COMMAS - ',
     *      /,1X,'THE I AND J NODES, ACTIVITY DURATION,'
     *      /,1X,'FIXED INTERRUPTION COST AND',
     *      /,1X,'VARIABLE INTERRUPTION COST FOR',
     *      /,1X,'EACH ACTIVITY.')
5003  FORMAT(/1X,'THE MINIMUM COMPLETION TIME FOR THIS',
     *' PROJECT IS:',2X,I6)
5004  FORMAT(/1X,'THE TOTAL COSTS OF THIS SCHEDULE:',
     *      /,1X,' VARIABLE INTERRUPTION COST..',I6,
     *      /,1X,' FIXED INTERRUPTION COST.....',I6,
     *      /,1X,' DELAY COST..................',I6,
     *      /,1X,'                              ------',
     *      /,1X,'           TOTAL COST.........',I6)
5005  FORMAT(/1X,'SELECT ONE OF THE FOLLOWING FUNCTIONS:',
     *      /,1X,'     (1) FIXED COST OF INTERRUPTION',
     *      /,1X,'     (2) VARIABLE COST OF INTERRUPTION',
     *      /,1X,'     (3) SUM OF FIXED AND VARIABLE COSTS',
     *      /,1X,'ENTER 1, 2, OR 3.')
5007  FORMAT(/1X,'YOU NOW HAVE THE FOLLOWING OPTIONS:',
     *      /,1X,'     (1) SELECT ACTIVITIES TO BE CONTINU',
     *      'OUS',
     *      /,1X,'     (2) VARY THE PROJECT COMPLETION TIME',
     *      /,1X,'     (3) STOP',
     *      /,1X,'ENTER 1, 2, OR 3')
```

```
5008  FORMAT(/1X,'HOW MANY ACTIVITIES DO YOU WISH TO',
     *' MAKE CONTINUOUS ?')
5009  FORMAT(/1X,'ENTER - SEPARATED BY COMMAS - THE',
     *      /,1X,'SEQUENTIAL ID NUMBERS OF THE',
     *      /,1X,'ACTIVITIES TO BE CONTINUOUS')
5010  FORMAT(/1X,'YOU NOW HAVE THE FOLLOWING OPTIONS:',
     *      /,1X,'     (1) ADD OR DELETE ACTIVITIES WHICH',
     *      /,1X,'         MUST BE CONTINUOUS',
     *      /,1X,'     (2) STOP',
     *      /,1X,'ENTER 1 OR 2')
5011  FORMAT(/1X,'ENTER THE NUMBER OF ACTIVITIES YOU WISH',
     *      /,1X,'TO REMOVE FROM CONTINUOUS SCHEDULING')
5012  FORMAT(/1X,'ENTER - SEPARATED BY COMMAS - THE ',
     *      /,1X,'SEQUENTIAL ID NUMBERS OF THE ACTIVITIES',
     *      /,1X,'WHICH ARE TO BE REMOVED')
5013  FORMAT(/1X,'ENTER A COMPLETION TIME GE ',I6,' AND LE',
     *I6)
5014  FORMAT(/1X,'YOU NOW HAVE THE FOLLOWING OPTIONS:',
     *      /,1X,'     (1)SELECT ANOTHER COMPLETION TIME',
     *      /,1X,'     (2) STOP',
     *      /,1X,'ENTER 1 OR 2')
5015  FORMAT(/1X,'THE MAXIMUM COMPLETION TIME IS ',I6)
5016  FORMAT(/1X,'WOULD YOU LIKE ANY INFO ON THE',
     *      /,1X,'SCHEDULED ACTIVITIES (YES/NO)')
5017  FORMAT(/1X,'YOU HAVE THE FOLLOWING OPTIONS:',
     *      /,1X,'     (1) LIST CONTINUOUS ACTIVITIES',
     *      /,1X,'     (2) LIST NON-CONTINUOUS ACTIVITIES',
     *      /,1X,'     (3) LIST INDIVIDUAL ACTIVITIES',
     *      /,1X,'     (4) CONTINUE SCHEDULING.',
     *      /,1X,'ENTER 1,2,3,OR 4')
C***
C***OBTAIN INPUT DATA
      WRITE(6,5000)
      READ(5,1001)YESNO
      IF(YESNO.NE.'YES') GO TO 99999
      WRITE(6,5001)
      READ(5,1000)NACTS,NCYC,DCOST
      WRITE(6,5005)
      READ(5,1000)YESNO
      TYPE=YESNO-1
      WRITE(6,5002)
      DO 10 N=1,NACTS
           READ(5,1000)I(N),J(N),D(N),ICOSTF(N),ICOSTV(N)
           II=I(N)
           JJ=J(N)
           SCNT(II)=SCNT(II)+1
           PCNT(JJ)=PCNT(JJ)+1
 10   CONTINUE
C***FIND MINIMUM COMPLETION TIME SCHEDULE AND COSTS
      DO 15 A=1,50
           PPCNT(A)=PCNT(A)
```

```
            PSCNT(A)=SCNT(A)
   15   CONTINUE
        CALL CYCLIC(S,MIND,TYPE,IND1,IND2,IND3,1)
        DO 17 A=1,50
             PCNT(A)=PDCNT(A)
             SCNT(A)=PSCNT(A)
   17   CONTINUE
C***WRITE MINIMUM COMPLETION TIME OUTPUT
        WRITE(6,5003)MIND
        DO 20 N=1,NACTS
             CALL LOWBND(N,S,LOW(N),LENGTH)
        ICOST1=ICOST1+LOW(N)*ICOSTF(N)
        ICOST2=ICOST2+LENGTH*ICOSTV(N)
   20   CONTINUE
        TCOST=ICOST1+ICOST2
        WRITE(6,5004)ICOST2,ICOST1,PDCOST,TCOST
C***ASK USER IF HE DESIRES ACTIVITY INFO
        WRITE(6,5016)
        READ(5,1001)YESNO
        IF(YESNO.NE.'YES') GO TO 30
   25   WRITE(6,5017)
        READ(5,1000)YESNO
        IF(YESNO.EQ.1) CALL WRITE1(S,LOW,$25)
        IF(YESNO.EQ.2) CALL WRITE2(S,LOW,$25)
        IF(YESNO.EQ.3) CALL WRITE3(S,$25)
C***FIND THE MAXIMUM COMPLETION TIME
   30   DO 31 N=1,NACTS
             IF(D(N).EQ.0) GO TO 31
             STRCNT=STRCNT+1
             STAR(N)='*'
   31   CONTINUE
        DMAX=0
        CALL CYCLIC(S,DMAX,TYPE,IND1,IND2,IND3,1)
        STRCNT=0
        DO 32 N=1,NACTS
             STAR(N)=' '
   32   CONTINUE
        DO 33 A=1,50
             PCNT(A)=PDCNT(A)
             SCNT(A)=PSCNT(A)
   33   CONTINUE
        WRITE(6,5015)DMAX
C***ALLOW USER TO SELECT OPTIONS
        WRITE(6,5007)
        READ(5,1000)YESNO
        IF(YESNO.EQ.3) GO TO 99999
        IF(YESNO.NE.1) GO TO 90
C***SELECT CONTINUOUS ACTIVITIES
   40   WRITE(6,5008)
        READ(5,1000)NUMB
        IF(NUMB.EQ.0) GO TO 55
```

```
      WRITE(6,5009)
      READ(5,1000)(LINE(K),K=1,NUMP)
      DO 50 K=1,NUMB
           IF(D(N).EQ.0) GO TO 50
           STRCNT=STRCNT+1
           N=LINE(K)
           STAR(N)='*'
 50   CONTINUE
C***SCHEDULE THE ACTIVITIES
 55   COMPT=0
      CALL CYCLIC(S,COMPT,TYPE,IND1,IND2,IND3,1)
      DO 65 A=1,50
           PCNT(A)=PPCNT(A)
           SCNT(A)=PSCNT(A)
 65   CONTINUE
C***WRITE CONTINUOUS SCHEDULE OUTPUT
      ICOST1=0
      ICOST2=0
      PDCOST=0
      TCOST=0
      WRITE(6,5003)COMPT
      DO 70 N=1,NACTS
           CALL FLOWBND(N,S,FLOW(N),LENGTH)
           ICOST1=ICOST1+FLOW(N)*ICOSTF(N)
           ICOST2=ICOST2+LENGTH*ICOSTV(N)
 70   CONTINUE
      PDCOST=(COMPT-MIND)*DCOST
      TCOST=PDCOST+ICOST1+ICOST2
      WRITE(6,5004)ICOST2,ICOST1,PDCOST,TCOST
C***ASK USER IF HE DESIRES ACTIVITY INFO
      WRITE(6,5016)
      READ(5,1001)YESNO
      IF(YESNO.NE.'YES') GO TO 77
 75   WRITE(6,5017)
      READ(5,1000)YESNO
      IF(YESNO.EQ.1) CALL WRITE1(S,FLOW,*75)
      IF(YESNO.EQ.2) CALL WRITE2(S,FLOW,*75)
      IF(YESNO.EQ.3) CALL WRITE3(S,*75)
C***SELECT THE NEW OPTIONS
 77   WRITE(6,5010)
      READ(5,1000)YESNO
      IF(YESNO.NE.1) GO TO 99999
      WRITE(6,5011)
      READ(5,1000)NUMB
      IF(NUMB.EQ.0) GO TO 40
      WRITE(6,5012)
      READ(5,1000)(LINE(K),K=1,NUMB)
      DO 80 K=1,NUMB
           IF(D(N).EQ.0) GO TO 80
           STRCNT=STRCNT-1
           N=LINE(K)
```

```
            STAR(I)=' '
   80   CONTINUE
        GO TO 60
C*********************************************************************
C***SELECT A NEW COMPLETION TIME
   90   WRITE(6,5013)MIND,DMAX
        READ(5,1000)MAXD
        IF(MAXD.GT.DMAX.OR.MAXD.LT.MIND) GO TO 90
C***SCHEDULE THE ACTIVITIES
        CALL CYCLIC(S,MAXD,TYPE,IND1,IND2,IND3,1)
        DO 97 A=1,50
             PCNT(A)=PDCNT(A)
             SCNT(A)=PSCNT(A)
   97   CONTINUE
C***WRITE THE OUTPUT
        ICOST1=0
        ICOST2=0
        PDCOST=0
        TCOST=0
        DO 100 N=1,NACTS
             CALL LOWBND(N,S,LOW(N),LENGTH)
             ICOST1=ICOST1+LOW(N)*ICOSTF(N)
             ICOST2=ICOST2+LENGTH*ICOSTV(N)
  100   CONTINUE
        PDCOST=(MAXD-MIND)*DCOST
        TCOST=PDCOST+ICOST1+ICOST2
        WRITE(6,5004)ICOST2,ICOST1,PDCOST,TCOST
C***ASK USER IF HE DESIRES ACTIVITY INFO
        WRITE(6,5016)
        READ(5,1001)YESNO
        IF(YESNO.NE.'YES') GO TO 112
  110   WRITE(6,5017)
        READ(5,1000)YESNO
        IF(YESNO.EQ.1) CALL WRITE1(S,LOW,$110)
        IF(YESNO.EQ.2) CALL WRITE2(S,LOW,$110)
        IF(YESNO.EQ.3) CALL WRITE3(S,*110)
C***SELECT THE NEW OPTIONS
  112   WRITE(6,5014)
        READ(5,1000)YESNO
        IF(YESNO.EQ.1) GO TO 90
 9999   STOP
C*********************************
        SUBROUTINE WRITE1(S,LOW,DUM)
        IMPLICIT INTEGER (A-Z)
C***
        COMMON /ONE/I(100),J(100)
        COMMON /TWO/NACTS,NCYC
        DIMENSION S(100,50),LOW(100)
C***
        WRITE(6,5050)
        M=0
```

```
      DO 10 N=1,NACTS
           IF(LOW(N).NE.0.OR.D(N).EQ.0) GO TO 10
           WRITE(6,5051)I(N),J(N),S(N,1)
   10 CONTINUE
 5050 FORMAT(1X,'CONTINUOUS ACTIVITIES ARE',
     *     /,3X,'I',2X,'J',1X,'S(N,1)')
 5051 FORMAT(2X,I2,1X,I2,3X,I4)
      RETURN 3
C*************************************
      SUBROUTINE WRITE2(S,LOW,DUM)
      IMPLICIT INTEGER (A-Z)
C***
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      DIMENSION S(100,50),LOW(100)
C***
      WRITE(6,5060)
      DO 10 N=1,NACTS
           IF(LOW(N).EQ.0.OR.D(N).EQ.0) GO TO 10
           WRITE(6,5061)I(N),J(N),(S(N,P),P=1,NCYC)
   10 CONTINUE
 5060 FORMAT(1X,'NON-CONTINUOUS ACTIVITIES ARE',
     *     /,4X,'I',4X,'J',1X,'SCHEDULE:')
 5061 FORMAT(10(1X,I4))
      RETURN 3
C*************************************
      SUBROUTINE WRITE3(S,DUM)
      IMPLICIT INTEGER (A-Z)
C***
      COMMON /ONE/I(100),J(100)
      COMMON /TWO/NACTS,NCYC
      COMMON /THREE/D(100)
      COMMON /SEVEN/ICOSTF(100),ICOSTV(100),DESC(100,4),DCOST
      DIMENSION S(100,50)
      WRITE(6,5070)
   10 READ(5,2000)A
 2000 FORMAT( )
      IF(A.GT.NACTS) GO TO 20
      WRITE(6,5071)I(A),J(A),D(A),ICOSTF(A),ICOSTV(A)
      WRITE(6,5072)
      WRITE(6,5073)(S(A,B),B=1,NCYC)
      GO TO 10
 5070 FORMAT(1X,'ENTER THE NUMBER OF THE ACTIVITY THAT YOU',
     *     ' WANT.',
     *     /,1X,'TO RETURN TO SCHEDULING, ENTER A NUMBER',
     *     /,1X,'GREATER THAN THE NUMBER OF ACTIVITIES')
 5071 FORMAT(1X,'I=',I2,' J=',I2,' D=',I2,' ICOSTF=',I2,
     *' ICOSTV=',I2)
 5072 FORMAT(1X,'THE SCHEDULE OF THE ACTIVITY IS:')
 5073 FORMAT(10(1X,I4))
   20 RETURN 2
      END
```

```
@XQT CPSS2.PRGM-D

 THIS PRØGRAM ALLØWS THE USER TØ EXAMINE
 RESULTING INTERRUPTIØN AND DELAY CØSTS
 FØR ALTERNATIVE SCHEDULES ØF A PARALLEL
 MULTI-CYCLE PRØJECT.
 DØ YØU HAVE A PRØBLEM TØ RUN ?
 (ENTER YES/NØ)
YES

 ENTER - IN INTEGER FØRMAT AND SEPARATED
 BY CØMMAS - THE NUMBER ØF ACTIVITIES,
 THE NUMBER ØF CYCLES IN THE PRØJECT,
 AND THE PER DAY PRØJECT DELAY CØST.
33,5,8

 SELECT ØNE ØF THE FØLLØWING FUNCTIØNS:
      (1) FIXED CØST ØF INTERRUPTIØN
      (2) VARIABLE CØST ØF INTERRUPTIØN
      (3) SUM ØF FIXED AND VARIABLE CØSTS
 ENTER 1, 2, ØR 3.
1

 ENTER - ØNE LINE AT A TIME, IN INTEGER
 FØRMAT, AND SEPARATED BY CØMMAS -
 THE I AND J NØDES, ACTIVITY DURATIØN,
 FIXED INTERRUPTIØN CØST AND
 VARIABLE INTERRUPTIØN CØST FØR
 EACH ACTIVITY.
1,2,4,2,2
2,3,2,3,3
3,4,14,1,1
4,5,2,4,4
5,6,5,1,1
5,7,1,3,3
5,10,10,1,1
5,11,3,5,5
5,12,10,3,3
5,14,16,2,2
6,8,5,2,2
7,9,2,2,2
8,10,5,2,2
8,14,5,2,2
9,11,0,0,0
9,14,0,0,0
10,12,5,3,3
11,15,3,2,2
12,13,3,4,4
13,14,3,3,3
14,15,3,4,4
```

```
15,16,5,1,1
15,17,3,3,3
16,17,0,0,0
16,18,0,0,0
16,19,3,2,2
17,19,5,2,2
18,19,5,1,1
19,20,0,0,0
19,21,5,1,1
20,21,5,3,3
21,22,3,5,5
22,23,4,2,2
 THE MINIMUM COMPLETION TIME FOR THIS PROJECT IS:      129.

 THE TOTAL COSTS OF THIS SCHEDULE:
  VARIABLE INTERRUPTION COST..    458
  FIXED INTERRUPTION COST.....     58
  DELAY COST..................      0
                                 ------
           TOTAL COST.........    516

 WOULD YOU LIKE ANY INFO ON THE
 SCHEDULED ACTIVITIES (YES/NO)
YES

 YOU HAVE THE FOLLOWING OPTIONS:
      (1) LIST CONTINUOUS ACTIVITIES
      (2) LIST NON-CONTINUOUS ACTIVITIES
      (3) LIST INDIVIDUAL ACTIVITIES
      (4) CONTINUE SCHEDULING.
 ENTER 1,2,3,OR 4
1
 CONTINUOUS ACTIVITIES ARE
   I   J S(N,1)
   1   2      0
   3   4      6
   5   7     79
   5  10     40
   5  11     79
   5  12     45
   5  14     24
   7   9     80
   8  14     71
  11  15     82
  22  23    109

 YOU HAVE THE FOLLOWING OPTIONS:
      (1) LIST CONTINUOUS ACTIVITIES
      (2) LIST NON-CONTINUOUS ACTIVITIES
      (3) LIST INDIVIDUAL ACTIVITIES
      (4) CONTINUE SCHEDULING.
```

```
 ENTER 1,2,3,ØR 4
2
 NØN-CØNTINUØUS ACTIVITIES ARE
    I     J SCHEDULE:
    2     3     4    14    16    18    20
    4     5    22    38    54    63    76
    5     6    50    55    60    65    78
    6     8    55    60    65    70    83
    8    10    60    65    70    75    88
   10    12    65    70    75    80    93
   12    13    76    79    82    85    98
   13    14    79    82    85    88   101
   14    15    82    85    88    91   104
   15    16    85    90    95   100   107
   15    17    87    90    93    96   107
   16    19    92    95   102   105   112
   17    19    90    95   100   105   112
   18    19    90    95   100   105   112
   19    21    95   100   105   110   117
   20    21    95   100   105   110   117
   21    22   106   109   112   115   122

 YØU HAVE THE FØLLØWING ØPTIØNS:
      (1) LIST CØNTINUØUS ACTIVITIES
      (2) LIST NØN-CØNTINUØUS ACTIVITIES
      (3) LIST INDIVIDUAL ACTIVITIES
      (4) CØNTINUE SCHEDULING.
 ENTER 1,2,3,ØR 4
4

 THE MAXIMUM CØMPLETIØN TIME IS    201

 YØU NØW HAVE THE FØLLØWING ØPTIØNS:
      (1) SELECT ACTIVITIES TØ BE CØNTINUØUS
      (2) VARY THE PRØJECT CØMPLETIØN TIME
      (3) STØP
 ENTER 1, 2, ØR 3
1

 HØW MANY ACTIVITIES DØ YØU WISH TØ MAKE CØNTINUØUS ?
1

 ENTER - SEPARATED BY CØMMAS - THE
 SEQUENTIAL ID NUMBERS ØF THE
 ACTIVITIES TØ BE CØNTINUØUS
4

 THE MINIMUM CØMPLETIØN TIME FØR THIS PRØJECT IS:     175

 THE TØTAL CØSTS ØF THIS SCHEDULE:
  VARIABLE INTERRUPTIØN CØST..   126
```

```
  FIXED INTERRUPTIØN CØST.....    23
  DELAY CØST..................   368
                               ------
           TØTAL CØST.........   517

 WØULD YØU LIKE ANY INFØ ØN THE
 SCHEDULED ACTIVITIES (YES/NØ)
NØ

 YØU NØW HAVE THE FØLLØWING ØPTIØNS:
      (1) ADD ØR DELETE ACTIVITIES WHICH
          MUST BE CØNTINUØUS
      (2) STØP
 ENTER 1 ØR 2
2

NØRMAL EXIT.  EXECUTIØN TIME:          2210 MLSEC.
```

```
@XQT CPSS2.PRGM-D

 THIS PRØGRAM ALLØWS THE USER TØ EXAMINE
 RESULTING INTERRUPTIØN AND DELAY CØSTS
 FØR ALTERNATIVE SCHEDULES ØF A PARALLEL
 MULTI-CYCLE PRØJECT.
 DØ YØU HAVE A PRØBLEM TØ RUN ?
 (ENTER YES/NØ)
YES

 ENTER - IN INTEGER FØRMAT AND SEPARATED
 BY CØMMAS - THE NUMBER ØF ACTIVITIES,
 THE NUMBER ØF CYCLES IN THE PRØJECT,
 AND THE PER DAY PRØJECT DELAY CØST.
33,5,8

 SELECT ØNE ØF THE FØLLØWING FUNCTIØNS:
      (1) FIXED CØST ØF INTERRUPTIØN
      (2) VARIABLE CØST ØF INTERRUPTIØN
      (3) SUM ØF FIXED AND VARIABLE CØSTS
 ENTER 1, 2, ØR 3.
1

 ENTER - ØNE LINE AT A TIME, IN INTEGER
 FØRMAT, AND SEPARATED BY CØMMAS -
 THE I AND J NØDES, ACTIVITY DURATIØN,
 FIXED INTERRUPTIØN CØST AND
 VARIABLE INTERRUPTIØN CØST FØR
 EACH ACTIVITY.
1,2,4,2,2
2,3,2,3,3
3,4,14,1,1
4,5,2,4,4
5,6,5,1,1
5,7,1,3,3
5,10,10,1,1
5,11,3,5,5
5,12,10,3,3
5,14,16,2,2
6,8,5,2,2
7,9,2,2,2
8,10,5,2,2
8,14,5,2,2
9,11,0,0,0
9,14,0,0,0
10,12,5,3,3
11,15,3,2,2
12,13,3,4,4
13,14,3,3,3
14,15,3,4,4
```

```
15,16,5,1,1
15,17,3,3,3
16,17,0,0,0
16,18,0,0,0
16,19,3,2,2
17,19,5,2,2
18,19,5,1,1
19,20,0,0,0
19,21,5,1,1
20,21,5,3,3
21,22,3,5,5
22,23,4,2,2
 THE MINIMUM CØMPLETIØN TIME FØR THIS PRØJECT IS:      129

 THE TØTAL CØSTS ØF THIS SCHEDULE:
  VARIABLE INTERRUPTIØN CØST..    458
  FIXED INTERRUPTIØN CØST.....     58
  DELAY CØST..................      0
                                ------
           TØTAL CØST.........    516

 WØULD YØU LIKE ANY INFØ ØN THE
 SCHEDULED ACTIVITIES (YES/NØ)
NØ

 THE MAXIMUM CØMPLETIØN TIME IS    201

 YØU NØW HAVE THE FØLLØWING ØPTIØNS:
      (1) SELECT ACTIVITIES TØ BE CØNTINUØUS
      (2) VARY THE PRØJECT CØMPLETIØN TIME
      (3) STØP
 ENTER 1, 2, ØR 3
2

 ENTER A CØMPLETIØN TIME GE    129 AND LE    201
175

 THE TØTAL CØSTS ØF THIS SCHEDULE:
  VARIABLE INTERRUPTIØN CØST..    104
  FIXED INTERRUPTIØN CØST.....      4
  DELAY CØST..................    368
                                ------
           TØTAL CØST.........    476

 WØULD YØU LIKE ANY INFØ ØN THE
 SCHEDULED ACTIVITIES (YES/NØ)
YES

 YØU HAVE THE FØLLØWING ØPTIØNS:
      (1) LIST CØNTINUØUS ACTIVITIES
      (2) LIST NØN-CØNTINUØUS ACTIVITIES
```

```
      (3) LIST INDIVIDUAL ACTIVITIES
      (4) CØNTINUE SCHEDULING.
 ENTER 1,2,3,ØR 4
3
 ENTER THE NUMBER ØF THE ACTIVITY THAT YØU WANT.
 TØ RETURN TØ SCHEDULING, ENTER A NUMBER
 GREATER THAN THE NUMBER ØF ACTIVITIES
26
 I=16 J=19 D= 3 ICØSTF= 2 ICØSTV= 2
 THE SCHEDULE ØF THE ACTIVITY IS:
  136  139  142  145  148
34

 YØU HAVE THE FØLLØWING ØPTIØNS:
      (1) LIST CØNTINUØUS ACTIVITIES
      (2) LIST NØN-CØNTINUØUS ACTIVITIES
      (3) LIST INDIVIDUAL ACTIVITIES
      (4) CØNTINUE SCHEDULING.
 ENTER 1,2,3,ØR 4
4

 YØU NØW HAVE THE FØLLØWING ØPTIØNS:
      (1)SELECT ANØTHER CØMPLETIØN TIME
      (2) STØP
 ENTER 1 ØR 2
2

NØRMAL EXIT.  EXECUTIØN TIME:          2463 MLSEC.
```

BIBLIOGRAPHY

1. Battersby, A., Network Analysis for Planning and Scheduling, Macmillan and Company, Ltd., London, 1967, 414 pages.

2. Burgess, A. R., and Killebrew, J. B., "Variation in Activity Level on a Cyclic Arrow Diagram," Journal of Industrial Engineering, Vol. 13, No. 2 (1962), pp. 76-83.

3. Burman, P. J., Precedence Networks for Project Planning and Control, McGraw-Hill, London, 1972, 374 pages.

4. Burney, S. M., A Technique for Scheduling Subcontracted Work in a Cyclic Project, M.S. Thesis, Georgia Institute of Technology, Atlanta, Georgia, 1971.

5. Ferraz, R. G., Continuity Considerations in Cyclic Project Scheduling, M.S. Thesis, Georgia Institute of Technology, Atlanta, Georgia, 1973.

6. Fisher, C., and Nemhauser, G. L., "Multicycle Project Planning," Journal of Industrial Engineering, Vol. 18, No. 4 (1967), pp. 278-284.

7. Moder, J. J., and Phillips, C. R., Project Management with CPM and PERT, Reinhold Publishing Corporation, New York, 1966, 283 pages.

8. Wiest, J. D., and Levy, F. K., A Management Guide to PERT/CPM, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1969, 170 pages.

9. Woodgate, H. S., Planning by Network: Project Planning and Control Using Network Techniques, Business Publications, London, 1967, 385 pages.